UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Redes neuronales profundas en energías renovables.

Autor: Eloy Anguiano Batanero
Tutor: José Dorronsoro Ibero

Junio 2018

Eloy Anguiano Batanero
Redes neuronales profundas en energías renovables.

Eloy Anguiano Batanero

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos.

“La ciencia será siempre una búsqueda,
jamás un descubrimiento real.
Es un viaje, nunca una llegada.”

Karl Raimund Popper

Agradecimientos

Me gustaría agradecer al Instituto de Ingeniería del Conocimiento y en particular a José el apoyo recibido tanto en efectos materiales como en paciencia en algunos casos.

También me gustaría dar las gracias al Centro Europeo de Previsiones Meteorológicas a Plazo Medio por la realización de un trabajo tan útil que ayuda a dar vida a proyectos que caminan hacia el progreso y a estudios científicos como éste.

Y por último, pero no por ello menos importante, gracias a mi familia y amigos que nunca faltaron cuando se les necesitaba tanto en los buenos momentos como en los malos.

Resumen

Hoy en día las energías renovables están adquiriendo cada vez más importancia ya que son imprescindibles a la hora de caminar hacia un sistema energético sostenible. La variación no controlada de la producción de éste tipo de energías gracias a factores no controlables por los humanos (por ejemplo, el clima) es uno de los factores más determinantes por el que no se consigue ese sistema energético completamente sostenible.

El camino hacia ese sistema pasa por, entre otras cosas, predecir la producción de energía que tendrá lugar en el futuro para, estimando una posible demanda, conocer si resultará suficiente como para intentar prescindir de energías no renovables.

Se ha visto que los paradigmas de aprendizaje automático actuales resultan muy útiles para hacer una estimación de una producción si se les proporcionan datos meteorológicos de calidad. Sin embargo, en el presente documento se ha tratado de averiguar qué horizontes temporales de predicción meteorológica resultarían más óptimos para el problema de la energía eólica.

Se han utilizado datos numéricos de predicciones meteorológicas así como técnicas tanto de regresiones simples como de regresiones más avanzadas a través de aprendizaje profundo para intentar encontrar un patrón de decisión sobre los mejores horizontes temporales de la predicción meteorológica. Este proceso se ha llevado a cabo en el lenguaje de programación *Python* y sus librerías para tratar archivos de predicción meteorológica (*Pygrib*) y para utilizar modelos de aprendizaje automático de manera sencilla (*Sklearn*).

Palabras clave

Redes Neuronales, Aprendizaje Automático, Predicción, Producción, Clima, Aprendizaje Profundo, Tiempo, Python, Sklearn

Abstract

Nowadays, renewable energies are becoming increasingly important as they are essential when it comes to moving towards a sustainable energy system. The uncontrolled variation of the production of this type of energy thanks to factors that cannot be controlled by humans (for example, weather) is one of the most determining factors that prevent this completely sustainable energy system from being so.

The way to achieve that system involves, among other things, predicting the future production of energy in order to estimate a possible demand and determine whether it will be sufficient to try to do without non-renewable energy.

Current automatic learning paradigms have proven to be very useful in estimating production if they are provided with quality meteorological data. However, this paper has attempted to find out which weather forecasting time horizons would be most optimal for the wind energy problem.

Numerical data from weather forecasts as well as techniques from both simple and more advanced regressions through in-depth learning have been used to try to find a decision pattern for the best weather forecast time horizons. This process has been carried out in the programming language *Python* and its libraries for processing weather forecast files (*Pygrib*) and for using automatic learning models in a simple way (*Sklearn*).

Keywords

Neural Networks, Machine Learning, Forecasting, Production, Climate, Deep Learning, Weather, Python, Sklearn

Índice

Listas

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

1.1. Objetivo

El objetivo del presente trabajo de fin de grado consiste en hacer un estudio sobre cómo influye la distancia temporal de una predicción meteorológica a la hora de predecir las producciones de energía eólica de una determinada región. En otras palabras: ¿Resulta siempre el modelo construido con los datos más a corto plazo el mejor para hacer predicciones más exactas ya que se supone un menor error asociado a la predicción, o por el contrario el modelo de predicción es más eficiente si trabaja con más días de antelación?. Para responder a esta pregunta debemos tener en cuenta que hemos escogido un modelo de predicción meteorológica de forma numérica que explicaremos más adelante.

También desarrollaré habilidades en análisis de datos, modelos predictivos, $machine\ learning$ y $Python$, que son habilidades bastante relevantes hoy en día gracias al auge del sector, tanto de forma empresarial como en innovación.

1.2. Motivación

Las energías renovables cada día tienen más relevancia a nivel mundial debido a las grandes contaminaciones generadas por las energías convencionales y la búsqueda de nuevos modelos sostenibles. Uno de los problemas asociados a la utilización exclusiva de energías renovables es la complicación de prever si la producción de las mismas será capaz de cubrir el gasto energético requerido ya que, de no hacerlo, habría que recurrir a las energías convencionales de cualquier modo.

Por tanto, la predicción de la producción (en éste caso eólica) es de vital importancia para contribuir a que el modelo sea sostenible y así poder intentar llegar a prescindir de las energías convencionales. Hoy en día ya existen modelos de predicción de la producción pero, ¿es posible que predicciones en principio más inexactas *a priori* puedan darnos más información y resultar más óptimas a la hora de saber cuánto se va a producir en un día o en determinado momento?

De esta pregunta nace el presente estudio, el cual se intentará abordar desde modelos básicos que,

si bien podrían sofisticarse para intentar minimizar al máximo el error de predicción y dar predicciones lo más exactas posibles, no es el objetivo del mismo, ya que solo se intenta proponer un acercamiento a la duda razonable anteriormente expuesta para una posible profundización posterior en el tema una vez obtenidos los resultados de este estudio.

1.3. Organización del documento

El presente documento se estructura de la siguiente manera:

Capítulo 2: Explicación del paradigma neuronal. Tipos de redes neuronales artificiales, explicación de aspectos clave para el $deep\ learning$ y estrategias a utilizar para conseguir resultados interesantes.

Capítulo 3: Introducción al funcionamiento del sistema atmosférico y sus complicaciones y explicación del modelo predictivo del clima a utilizar.

Capítulo 4: Descripción del método experimental seguido y análisis de los datos obtenidos a través del mismo.

Capítulo 5: Conclusiones generales del estudio y posibles líneas de trabajo a seguir en caso de querer continuar con el mismo.

2

Redes Neuronales

2.1. Definición del paradigma neuronal

Las redes neuronales, el algoritmo de predicción que usaremos más adelante, son un método de resolución de problemas "bioinspirado". Es importante conocer las bases biológicas sobre las que se apoya el paradigma antes de tratar la abstracción utilizada en el mundo de la informática teórica. Por esta razón, procederé a la explicación de las bases de las neuronas biológicas y sus componentes para identificar las partes de una neurona artificial y qué función le compete a cada una de ellas en el modelo original.

2.1.1. La neurona biológica

En 1906 Ramón y Cajal ganó el Premio Nobel de Medicina y Fisiología por su trabajo de investigación de la estructura del sistema nervioso con Camillo Golgi de manera conjunta. Ramón y Cajal propuso la *"Doctrina de la Neurona"* [1], según la cual se establece la neurona como célula básica de operación en el sistema nervioso y, por tanto, de las redes neuronales biológicas.

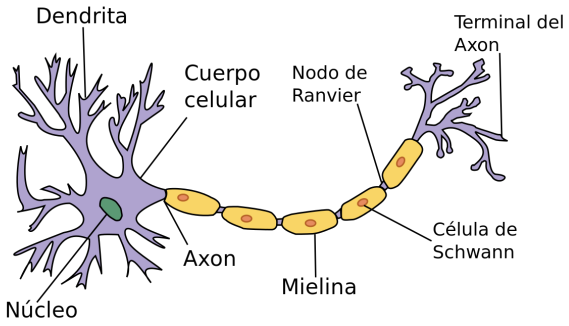


Figura 2.1: Imagen esquemática de una neurona biológica. (Imagen extraída de Wikipedia [2]).

Las neuronas establecen circuitos de potenciales eléctricos a través de la excitabilidad de sus membranas. Esta excitabilidad depende de la conductancia establecida por los puentes sinápticos de dichas neuronas, las cuales pueden servir de entrada a otras neuronas y así, sucesivamente, se irá

conformando una red neuronal.

Pese a que existan distintos tipos de neuronas biológicas (según el propósito que tengan las mismas), todas comparten una serie de elementos comunes.

Sinápsis: Es un proceso de conexión entre dos neuronas mediante el cual una libera cierto tipo de neurotrasmisores con el objetivo de producir una reacción a las neuronas colindantes. Esta liberación de neurotrasmisores hará que la conexión sináptica entre dos neuronas determinadas pueda verse reforzada o mermada según interese.

Dendritas: Son las prolongaciones de la neurona post-sináptica encargadas de recibir los estímulos de neurotrasmisores producidos por distintas neuronas pre-sinápticas.

Soma: Se trata del cuerpo celular de la neurona. Su membrana es una capa lipídica que es capaz de propagar potenciales eléctricos según lo recibido a través de las dendritas.

Axón: Se trata de la otra prolongación de la parte distal de la neurona por la cual se liberan los neurotrasmisores al espacio sináptico para ser recogidas por las nuevas neuronas post-sinápticas.

En el cerebro humano hay alrededor de cien mil millones de neuronas con un promedio de $7{,}000$ conexiones sinápticas por neurona. Las conexiones y topología de las redes conformadas por esas neuronas son las que nos permiten desarrollar nuestro intelecto y llegar a abstraernos y aprender del mundo que nos rodea.

2.1.2. La neurona artificial

En 1943, Warren Sturgis McCulloch y Walter Harry Pitts propusieron un modelo de neurona artificial (la neurona de McCulloch-Pitts) [3] que intentaba simular el comportamiento de una neurona biológica, simplificando la misma para acomodarlo a un modelo simple pero compartiendo caracteristicas.

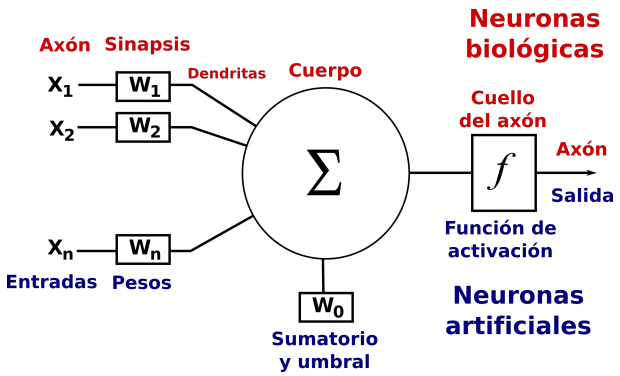


Figura 2.2: Comparación entre una neurona biológica y una artificial. (Imagen extraída del Blog de Fernando Sancho Caparrini [4]).

En la figura 2.2 podemos observar que existen las mismas partes de la neurona biológica simplificadas o modeladas de tal forma que son lo suficientemente funcionales para la aproximación en cuestión pero no tan complejas como el sistema biológico real. Homólogamente a la neurona biológica, en la neurona artificial distinguimos:

Vector de pesos de conexiones (Sinapsis): Es una colección de valores que simboliza los pesos de las conexiones entre neuronas de la forma $W = \{W_{11}, W_{12}, ...W_{ij}\}$ donde W_{ij} es el peso de la conexión entre la neurona pre-sináptica i y la neurona post-sináptica j.

Vector de valores de entrada (Dendritas): Se trata de otra colección de valores que corresponden a los valores de entrada de cada una de las dendritas de la neurona $X = X_i$.

Función de propagación (Soma): Toda neurona necesita de una función que aúne la información recibida por sus dendritas para que el resultado de esa junto con el valor umbral del potencial eléctrico de la membrana volviendo al modelo biológico, o simplemente el valor umbral (w_0) de la misma en el modelo artificial, pueda ser tratado posteriormente por la función de activación.

$$y_j = w_0 + \sum_{i=1}^{N} x_i * w_{ij}. \tag{2.1}$$

Cabe destacar que aunque se haya presentado el sumatorio como la función de propagación de la neurona artificial, existen otro tipo de funciones como, por ejemplo, el productorio:

$$y_j = w_0 + \prod_{i=1}^{N} x_i * w_{ij}. \tag{2.2}$$

Función de activación (Axón) Esta función es la que se encarga de establecer el estado o la cuantía de excitación que emitirá la neurona pre-sináptica hacia las nuevas neuronas post-sinápticas. En el caso de que escojamos la función de propagación del sumatorio anteriormente propuesta, quedaría de la forma:

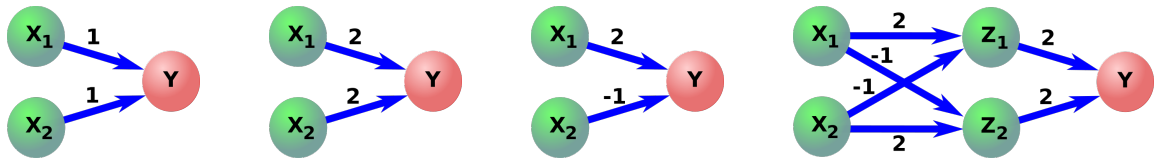
$$f(y_j) = f(w_0 + \sum_{i=1}^{N} x_i * w_{ij}). \tag{2.3}$$

En el apartado 2.3 se tratará este tipo de funciones más en detalle.

2.1.3. Redes de McCulloch-Pitts

Las redes de McCulloch-Pitts son redes de neuronas con una función de activación binaria y un umbral a partir del cual establecer su estado de activación/desactivación. Combinando valores de

pesos entre las conexiones de las neuronas y los umbrales de activación de las neuronas se obtienen redes especialmente útiles para modelar funciones lógicas. Este tipo de redes neuronales es el primer acercamiento al modelo que conocemos en la actualidad de las mismas. Los pesos los establece el diseñador y son estáticos, por lo que aún no podemos hablar de una red neuronal que aprenda, como expresa Jürgen Schmidhuber en su artículo *"Deep Learning in Neural Networks: An Overview"* [5].



(a) Función AND (b) Función OR (c) Función ANDNOT (d) Función XOR

Figura 2.3: Funciones lógicas básicas con redes de McCulloch-Pitts.

Al establecer un valor umbral de 2 en todas las neuronas que reciben entradas para determinar su activación binaria, podemos ver como en la figura 2.3 se modelan distintas funciones lógicas a través de este tipo de redes.

2.1.4. Redes neuronales como funciones de regresión avanzadas

Pese a que las redes neuronales se pueden utilizar para otros tipos de problemas como pueden ser problemas de clasificación, reducción dimensional o $clustering$, nos centraremos en la funcionalidad de regresión que puede ofrecernos el modelo de redes neuronales.

Regresión lineal

Una regresión lineal sobre un conjunto de datos se trata de un modelo matemático que nos aporta una la aproximación lineal para establecer la relación de dependencia entre dos o más variables.

En el caso particular de dos variables (por comodidad de representación) la forma de obtener esa expresión sería trazando una recta que minimice el error cuadrático medio del conjunto de puntos. De esta forma podremos obtener una función aproximada de la distribución de la forma

$$f(x) = mx + c, \tag{2.4}$$

donde m es la pendiente de la recta y c el desplazamiento sobre el origen.

Regresion no lineal

Como es de suponer, no todos los conjuntos de datos tienen una aproximación lineal lo suficientemente válida como para que sea correcto modelizarlas a través de la regresión lineal, por lo que existen

otro tipo de regresiones no lineales que utilizan distintas estrategias. Estas pueden ser polinomios de distintos grados (Ej. $f(x) = ax^2 + bx + c$) o funciones matemáticas no lineales en general (exponenciales, logarítmicas, etc ...) con el fin de establecer la mejor curva de ajuste en base a la minimización de algún tipo de función de error, tema que trataremos en el apartado 2.4 de este documento.

Regresión en las redes neuronales

Como hemos comenzado explicando, las redes neuronales son un sistema especialmente bueno para establecer una función de regresión a una muestra de datos. El problema a resolver trata de predecir uno o más valores a partir de unos de entrada. A través del aprendizaje de las redes neuronales que trataremos en el apartado 2.5, la red neuronal es capaz de establecer una configuración de su vector de pesos en base a la minimización de un error, temas que trataremos más en profundidad en los apartados 2.6 y 2.4.

La complicación a la hora de resolver un problema de regresión con redes neuronales está en escoger de forma efectiva cómo obtener y optimizar el error, ya que esto será determinante a la hora de obtener una solución aceptable al problema y, sobre todo, en cómo regularizar ese aprendizaje de forma que sea lo suficientemente genérico y no acabar sobreajustando la red a los ejemplos que le pasemos al entrenamiento de nuestra red como veremos en el apartado 2.7.

2.2. Arquitectura

Escoger la arquitectura o la topología de una red neuronal es una de las tareas más complicadas a la hora de solucionar un problema. Hay múltiples clasificaciones de las mismas: según el número de capas (monocapa, multicapa o profundas), según el tipo de conexiones (recurrentes y no recurrentes) o según el grado de conexión (parcial o totalmente conectadas).

Debido a la gran combinación de topologías existentes, nos centraremos en explicar tan solo tres tipos de redes neuronales que se han mostrado como bastante resolutivas para distintos tipos de problemas y profundizaremos en las redes totalmente conectadas ya que son las que utilizaremos para la resolución de nuestro problema en particular.

2.2.1. Redes totalmente conectadas (*"Fully Connected"*)

Las redes neuronales totalmente conectadas son aquellas en las que cada neurona de la capa anterior a la observada está conectada con todas las neuronas de la capa siguiente. De esta forma se consigue no condicionar con el diseño, ya que no conectar dos neuronas es equivalente a especificar que el peso entre ambas ha de ser siempre nulo, y permite que el algoritmo de aprendizaje sea capaz de sacar mejor sus propias conclusiones del conjunto de datos de entrenamiento.

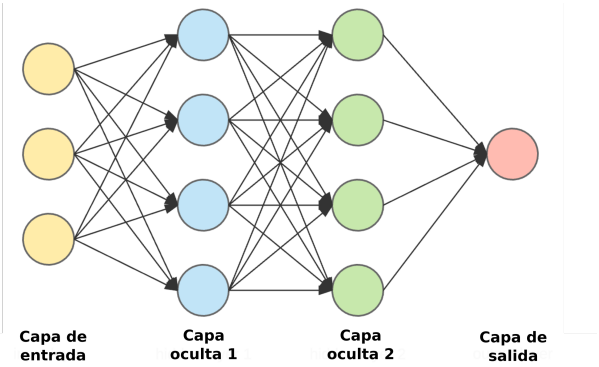


Figura 2.4: Red totalmente conectada y multicapa (Imagen extraída de la Web *"Towards Data Science"* [6])

2.2.2. Redes convolucionales

La peculiaridad de las redes convolucionales reside en que sus neuronas operan con matrices de valores en lugar de con valores únicamente numéricos. Este tipo de redes son realmente útiles para el tratamiento de imágenes, especialmente desde que fueron optimizadas en el año 2012 por Dan Ciresan [7].

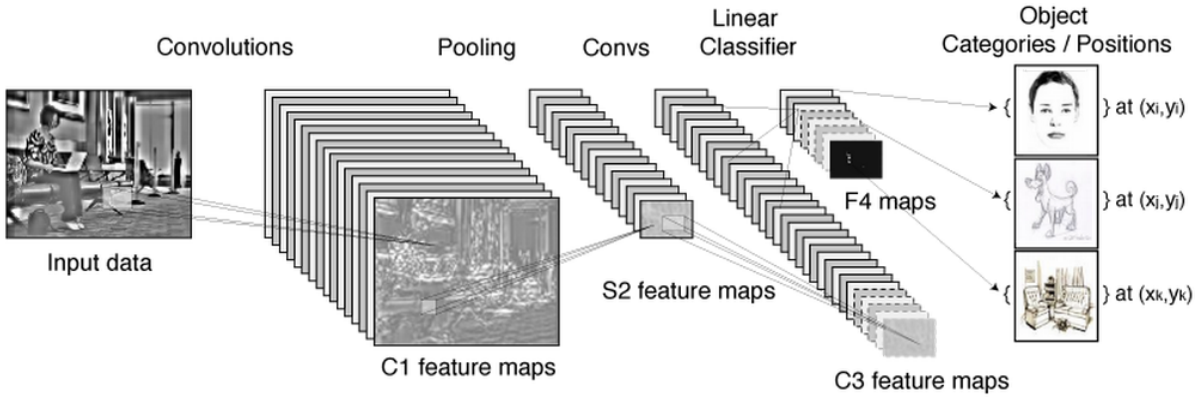


Figura 2.5: Red convolucional con varias capas convolucionales. (Imagen extraída del Blog de "*Kurzweil accelerating intelligence*" [8])

Su estrategia reside en procesar progresivamente problemas más pequeños para tratarlos en una red más profunda y después realizar una reducción o *pooling* con el objetivo de reducir el sobreajuste de la red al problema concreto, tema que se tratará en el apartado 2.7 del presente documento.

2.2.3. Redes recurrentes

Este tipo de redes neuronales se caracteriza por tener ciclos entre neuronas, es decir, hay neuronas (o capas enteras) cuya salida irá a parar a neuronas de capas anteriores o de su misma capa.

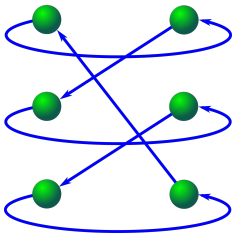


Figura 2.6: Red neuronal recurrente esquemática (Imagen inspirada de la Web de StackExchange [9]).

La fortaleza de este modelo reside en tener un recordatorio de algún estado anterior de la red. Por ello, serán un modelo a valorar especialmente cuando la salida del sistema dependa de estados anteriores del mismo. Un ejemplo bastante significativo y metafórico podemos ser nosotros mismos. Siempre que afrontamos un problema lo situamos y lo evaluamos según un contexto anterior (secuencias de imágenes, procesamiento del lenguaje natural, etc...) y no desde cero.

2.3. Función de activación

En el paradigma de las redes neuronales es necesario establecer una función de activación de cada neurona, por la cual se establece bajo qué condiciones de entrada a la neurona, la neurona responderá y de qué manera lo hará. La elección de dicha función de activación de la neurona es un punto realmente importante para el modelo ya que, dependiendo del tipo de problema a resolver y de la arquitectura o topología de tu red neuronal existen funciones de activación que hacen que la solución al problema sea lo suficientemente correcta o pase a ser completamente inservible. Las funciones más extendidas son y se pueden ver en la figura 2.7:

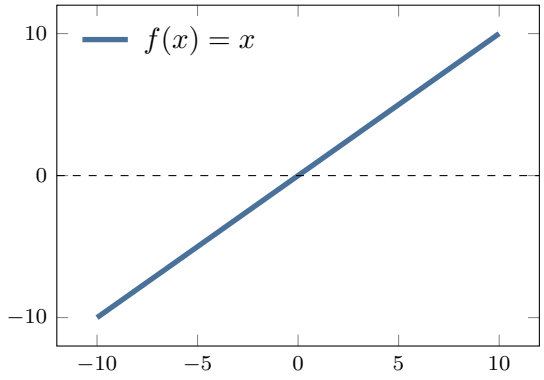
Función identidad La salida es igual a la entrada, por lo que se trata de una función de activación bastante sencilla que no aplica ninguna transformación a la información de entrada.

Función logística Se trata de una función sigmoide que consigue dar un valor continuo a la activación de la neurona y se trata de algo más acotado comparado con la función anterior. La función es diferenciable en todo su dominio.

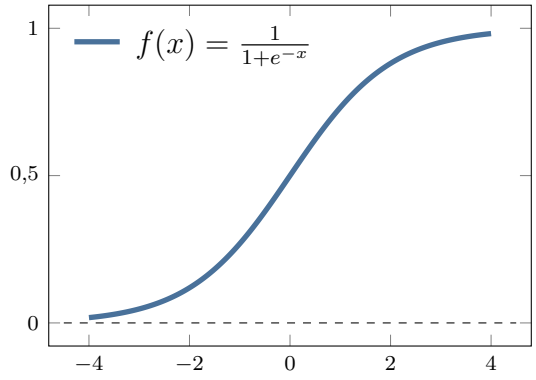
Función tangente hiperbólica Consigue mantener la suavización de la función anterior pero estableciendo el rango a $(-1, 1)$. La función es completamente diferenciable.

Función escalón Define un estado booleano de la neurona (ON, OFF), activándose si la información de entrada (junto con sus correspondientes pesos) es superior a un umbral(θ) dado. A pesar de que la función es diferenciable en todo su dominio excepto en el punto $x = \theta$, su derivada es 0, cualidad que dificulta en gran medida el aprendizaje por retropropagación explicado en el apartado 2.5.1.

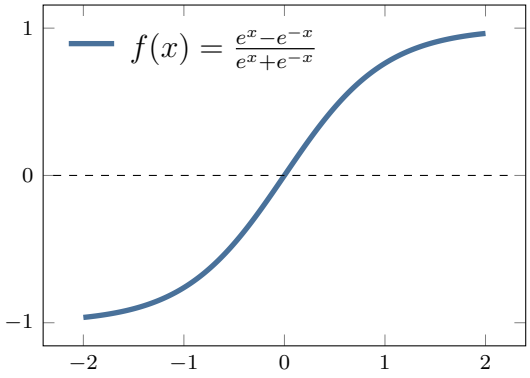
Función lineal rectificadora (*"ReLU"*) Se trata de una simplificación de las funciones ante-



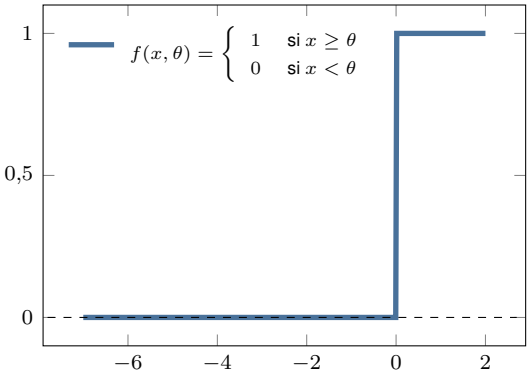
(a) Función identidad.



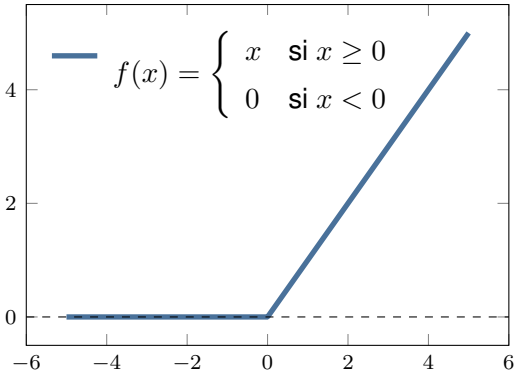
(b) Función logística.



(c) Función tangente hiperbólica.



(d) Función escalón ($\theta = 0$).



(e) Función lineal rectificadora "$ReLU$".

Figura 2.7: Distintas funciones de activación.

riores que otorga serias ventajas computacionales al entrenamiento de cualquier red. La función es completamente diferenciable en todo su dominio excepto en el punto $x = 0$ y su derivada es 1 en el rango $(0, +\infty)$ y 0 en el rango $(-\infty, 0)$.

2.4. Funciones de error

Como hemos comentado anteriormente en el apartado 2.1.4, las redes neuronales pueden ser utilizadas en un problema de regresión avanzado. Por tanto, es necesario establecer una función de error para verificar qué curva de ajuste es la más optima para modelar el problema a tratar.

Buscamos una solución aproximada a un problema concreto de la forma $t = f(x)$. Hemos de minimizar el error entre y (el resultado obtenido) y t (el resultado correcto o $target$). En algunos casos ese error nos interesa que sea de una determinada forma para hacer que nuestra red neuronal evalúe su error de una forma más conveniente u óptima computacionalmente hablando. A pesar de que existan más funciones de error para evaluar la efectividad de una red neuronal, nos centraremos en la métrica que utilizaremos en este caso, que es el error cuadrático medio. Otras funciones de error serían el *Mean absolute Error* (MAE) y el *Root Mean Squared Error* (RMSE) :

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|t_i - y_i| \qquad RMSE = \frac{1}{N}\sqrt{\sum_{i=1}^{N}(t_i - y_i)^2} \tag{2.5}$$

Error Cuadrático Medio ("ECM")

El *Error Cuadrático Medio* (ECM) se define de la forma

$$ECM = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2. \tag{2.6}$$

Podemos observar que el error cuadrático medio es la media de los cuadrados (para evitar influencias de signos) de la diferencia entre el valor esperado y el obtenido. Pero ¿por qué no coger el valor absoluto en vez de los valores cuadráticos? La respuesta a esta pregunta es que la función cuadrática es más sencilla de manejar matemáticamente. Por ejemplo, la función del ECM es derivable en todos sus puntos (propiedad muy interesante ya que es una función que debemos minimizar) mientras que la del valor absoluto no.

2.5. Aprendizaje en las redes neuronales

El aprendizaje de una red neuronal consiste en actualizar las matrices de pesos entre las capas de la red neuronal para conseguir minimizar la función de error. La minimización de la función será calculada a través de descenso por gradiente. Por ello, para cada capa, la actualización de pesos W_n será calculada de la siguiente forma sucesiva

$$W_n = W_{n-1} - \lambda \overrightarrow{\nabla E} \tag{2.7}$$

siendo λ la tasa de aprendizaje de la red, es decir, la magnitud del paso que se da en la dirección indicada por el gradiente del error $\overrightarrow{\nabla E}$ calculado mediante el algoritmo de $"backpropagation"$ explicado en el apartado 2.5.1.

2.5.1. Algoritmo de *"backpropagation"*

El algoritmo de $"backpropagation"$ o retropropagación [10] es el algoritmo que nos permite calcular el gradiente del error capa a capa de la red neuronal ($\overrightarrow{\nabla E}$) y, de esta forma, actualizar todos los pesos correspondientes de la misma para conseguir minimizar esa función de error, lo que se entiende como "aprender".

El algoritmo de retropropagación se basa en que, partiendo de un error obtenido con la comparación de la salida obtenida y y el valor objetivo t se pueda establecer el error fraccional de cada neurona de capas anteriores. Esto nos permite averiguar la fracción de error provocada por cada camino individual desde la capa de salida hasta la entrada y poder así, posteriormente, actualizar los pesos de manera que se minimice la función de error elegida.

Si bien el algoritmo de retropropagación se encarga de calcular ese gradiente del error ($\overrightarrow{\nabla E}$) desde la capa de salida hacia la capa de entrada, al ir siempre acompañado de evaluar la salida actual (paso anterior de $feedforward$) y de actualizar los pesos (paso posterior), hay casos en los que nos podremos referir a éste como un algoritmo de aprendizaje en general y no sólo como un método de cálculo del gradiente del error .

2.5.2. Tipos de aprendizaje

Una vez conocemos lo que significa aprender (actualizar los pesos de conexiones entre neuronas), podemos distinguir varios tipos de aprendizaje según los procesos que siguen los mismos.

Aprendizaje supervisado

Es aquel aprendizaje al que se le presentan unos ejemplos previamente evaluados de los patrones a tratar, es decir, que ya se conoce la salida esperada para una cantidad del conjunto de datos.

Según dicha salida esperada, podemos distinguir dos tipos de problemas: regresión y clasificación. Un problema de regresión intenta establecer una función de ajuste a los datos para predecir uno o más valores continuos en función de otros dados, mientras que un problema de clasificación discrimina de forma discreta entre dos o más clases según unos parámetros dados.

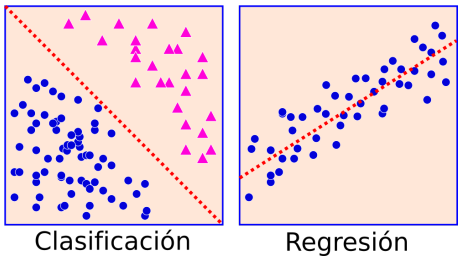


Figura 2.8: Comparación entre un problema de regresión y uno de clasificación. (Imagen creada a partir de imagen extraída de la Web de introducción al $machine\ learning$. [11])

Los problemas de clasificación se pueden ver como encontrar la función que ajuste la mejor frontera de separación entre los datos y, de esa forma, ser capaz de clasificar mientras que en los problemas de regresión la función de ajuste es el objetivo del mismo.

Aprendizaje no supervisado

El aprendizaje no supervisado es aquel en el cual no tenemos un grupo de datos previamente evaluados (ya bien sea en clases o en un número escalar). Por ello, hemos de encontrar herramientas que nos permitan la extracción de información desde unos datos de los cuales no tenemos ninguna información previa, como pueden ser el $clustering$ o la reducción de la dimensionalidad.

2.6. Optimización de la función de error

Como hemos explicado anteriormente, aprender significa minimizar una función de error, por lo que necesitaremos distintos algoritmos de optimización. La libreria Sklearn de *Python* nos ofrece los siguientes optimizadores: *Stochastic Gradient Descent* (SGD) , *ADAptative Moment estimation* (ADAM) y *Limited-memory BFGS* (LBFGS) .

2.6.1. SGD(*"Stochastic Gradient Descent"*)

Utiliza el método de descenso por gradiente anteriormente expuesto en el apartado 2.5.1 solo que la elección de los ejemplos se hace mediante un *shuffle* previo de los datos. Se distingue dos formas bajo las que puede trabajar este optimizador.

Mini-Batch: Los datos de entrenamiento se dividen en grupos de N elementos y el cálculo del error se hace con la media del error de esos N elementos junto con la actualización de los pesos.

Online: El cálculo del error y la actualización de los pesos tiene lugar con cada presentación de un nuevo dato a la red.

2.6.2. ADAM ("ADAptative Moment estimation")

ADAM fue presentado en 2015 por Diederik Kingma y Jimmy Ba en un artículo científico llamado "Adam: A Method for Stochastic Optimization" [12]. Sus autores definían el algoritmo como la combinación de las fortalezas entre dos algoritmos anteriores: *Adaptative Gradient Algorithm* (AdaGrad) y *Root Mean Square Propagation* (RMSProp) .

Para acercarnos al algoritmo ADAM hemos de añadirle un nuevo parámetro a la explicación del descenso por gradiente anteriormente explicado: el momento. El momento responde a la aceleración del descenso por gradiente. La expresión de la actualización de los pesos según este optimizador sería

$$W_n = W_{n-1} + \eta \Delta W_{n-1} - \lambda \overrightarrow{\nabla E}, \tag{2.8}$$

siendo $\Delta W_{n-1} = W_{n-1} - W_{n-2}$ y η la tasa del momento que le da importancia al cambio de pesos en la iteración anterior.

Pero el optimizador ADAM no se basa sólo en esta premisa, sino que es capaz de actualizar λ y η mediante una serie de parámetros extra. En total, los parámetros necesarios para que este optimizador funcione serían los siguientes:

- β_1 Tasa de decrecimiento exponencial del gradiente para la estimación del momento.
- β_2 Tasa de decrecimiento exponencial del gradiente al cuadrado para la estimación del momento.
- λ Tasa de aprendizaje o "$Step\ size$"
- ϵ Constante de prevención de la división por cero.

2.6.3. LBFGS ("Limited-memory BFGS")

Utiliza métodos quasi-newtonianos como alternativa a los newtonianos por complicaciones computacionales a la hora de calcular matrices Hessianas para encontrar los máximos o los mínimos de la función de error, por lo que utiliza aproximaciones a las mismas.

2.7. Regularización

En la sección de aprendizaje 2.5 decimos que las redes neuronales son una herramienta muy potente para encontrar una función de regresión que modele los datos presentados pero ¿y si son demasiado potentes?. Aquí es donde aparece el problema del sobreajuste u "$overfitting$". El sobreajuste es el efecto de ajustar demasiado tu función a los datos de entrenamiento presentados, lo cual puede extraer relaciones no causales que merman la capacidad de generalizar de la red neuronal.

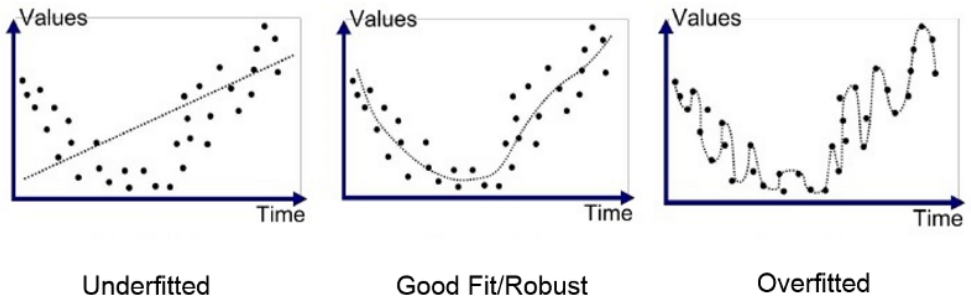


Figura 2.9: Ejemplo de "$underfitting$" (imagen de la izquierda) y "$overfitting$" (imagen de la derecha) en comparativa con un modelo correctamente ajustado (imagen central). (Imagen extraída de Foro sobre *Machine Learning* [13]).

Una red neuronal ha de ser capaz también de generalizar lo bastante como para encontrar la función más sencilla que modele lo suficientemente bien los datos presentados. Por ello, hemos de establecer una penalización al crecimiento de los coeficientes de la función de ajuste para alcanzar el equilibrio y encontrar una función de ajuste capaz de generalizar y apropiada para el problema en cuestión.

Mediante el método de "$L2\ penalty$" le añadiremos una penalización al resultado de la función de error de la propia red neuronal con mayores coeficientes en la función de ajuste:

$$E_{L2} = Error + \alpha \left\| W \right\|^2 , \tag{2.9}$$

donde $\|W\|$ es la norma de la matriz de pesos W y α es la tasa de penalización que se quiere aplicar.

2.8. Inicialización.

La minimización de una función de error descendiendo por el gradiente siempre tiene el problema de que dependiendo de los valores de inicio podemos vernos atrapados en un mínimo local y no en la optimización completa del problema alcanzando un mínimo cercano al absoluto de la función de error.

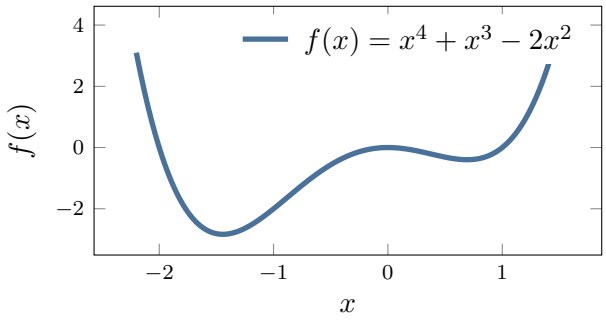


Figura 2.10: Función $f(x) = x^4 + x^3 - 2x^2$ con un mínimo global en el intevalo $[-2, -1]$ y un mínimo local en el intervalo $[0, 1)$

Representando este problema en dos dimensiones para hacerlo visible tenemos la figura 2.10. En ella podemos observar que haciendo un descenso por gradiente partiendo de cualquier punto en el intervalo $(0, \infty)$ llegaremos al resultado del mínimo local, mientras que partiendo de un punto en el intervalo $(-\infty, 0)$ conseguiremos llegar al mínimo absoluto.

Si los pesos iniciales de una red neuronal son demasiado bajos, las señales acaban siendo demasiado pequeñas al pasar a través de las capas que dejan de ser útiles. Es por eso que debemos encontrar el punto en el que sepamos que la red neuronal es capaz de trabajar con esos pesos.

Inicializar correctamente o no una red neuronal afectará en una gran medida a la eficacia con la que esta red consiga establecer una buena función de ajuste al problema entre manos y, por tanto, se verá enormemente reflejado este hecho en la utilidad de la red para la predicción de dicho problema. Es por esto que se debe escoger algún tipo de inicialización que te ofrezca según qué ventajas para el problema a tratar.

Inicialización de Xavier

En su artículo *"Understanding the difficulty of training deep feedforward neural networks"* [14], Xavier Glorot y Yoshua Bengio presentan su método particular de inicialización de pesos de una red neuronal. Su método se basa en conseguir establecer una normalización de los pesos y conseguir con ellos una forma de Gaussiana normalizada para conseguir que la actualización de pesos se realice en todas las capas, ya que un gradiente de error nulo significa que ese peso no será actualizado.

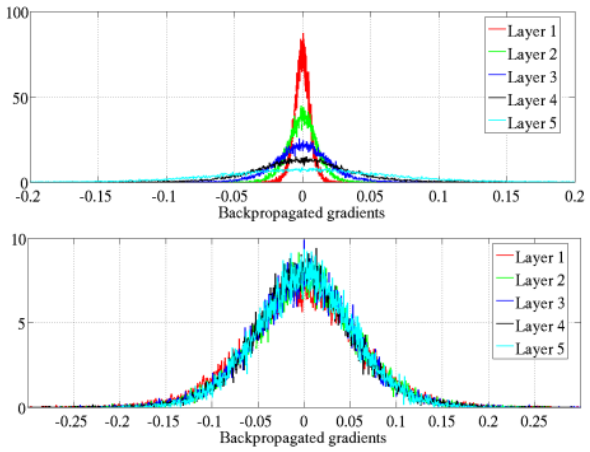
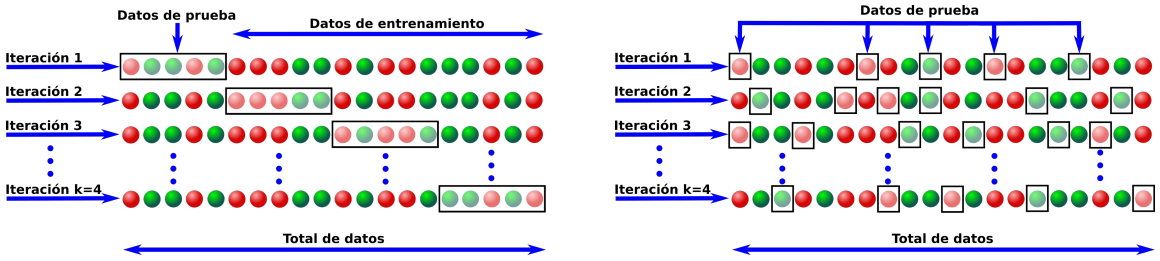


Figura 2.11: Histograma de gradientes por capa tras distintas inicializaciones. Inicialización heurística aleatoria (imagen superior) VS Inicialización de Xavier (imagen inferior). (Imagen extraída de la explicación del artículo [15]).

Así obtenemos una distribución normal de los pesos de todas las capas, lo que nos ayuda a que la actualización de los pesos se haga de igual forma durante todas las capas y no en unas en mayor medida que otras.

2.9. Validación cruzada e hiperparametrización

La validación cruzada se trata de una estrategia de comprobación de cuán generalista y cuán útil es una red neuronal a través de intentar establecer una independencia de los subconjuntos de datos de entrenamiento ($train$) y los de validación escogidos con el fin de suavizar anomalías estadísticas de esos subconjuntos. Hay dos tipos de validaciones cruzadas:



(a) K-fold cross validation

(b) Random cross validation

Figura 2.12: Validación cruzada de K hojas con $K = 4$ VS validación cruzada aleatoria.(Imagen basada en imagen extraída de Wikipedia [16]).

Validación cruzada de K hojas Se dividen los datos en K bloques de elementos. A partir de este momento el problema trabaja estableciendo cada una de esas divisiones como el conjunto de test y el resto como el conjunto de entrenamiento. Así, el error de la red

neuronal será calculado como la media aritmética de los errores individuales de cada una de las K hojas: $Error = \frac{1}{k}\sum_{i=1}^{K} Error_i$

Validación cruzada aleatoria Se hacen K iteraciones de validación escogiendo en cada una de ellas N elementos de forma aleatoria que conformarán el conjunto de test mientras que el resto conformarán el grupo de $train$. La principal diferencia con la validación anterior reside en que un mismo dato puede ser utilizado varias veces en el grupo de test, mientras que en el caso anterior no, por lo que ésta no suele ser una de las mejores opciones a utilizar.

Por otro lado tenemos la hiperparametrización, pero primero debemos preguntarnos: ¿qué es un hiperparámetro? Un hiperparámetro es aquel que no puede ser aprendido a través del proceso de entrenamiento de la red neuronal, es decir, es un parámetro de más alto nivel que el diseñador de la red neuronal escoge en un momento dado.

Conociendo de qué se trata un hiperparámetro debemos entrar en el proceso de hiperparametrización, que no es otro que escoger de forma adecuada estos parámetros para el problema que se vaya a tratar. Este proceso consiste en hacer un estudio del comportamiento de la red neuronal estableciendo el/los hiperparámetros como variables independientes e intentar encontrar el punto donde se maximiza la validez de la función de ajuste establecida.

Normalmente se trata de un proceso muy costoso, por lo que se evitará hacerlo con muchos de los parámetros de la red por no aumentar la dimensionalidad del espacio de búsqueda y aumentar considerablemente los casos de prueba.

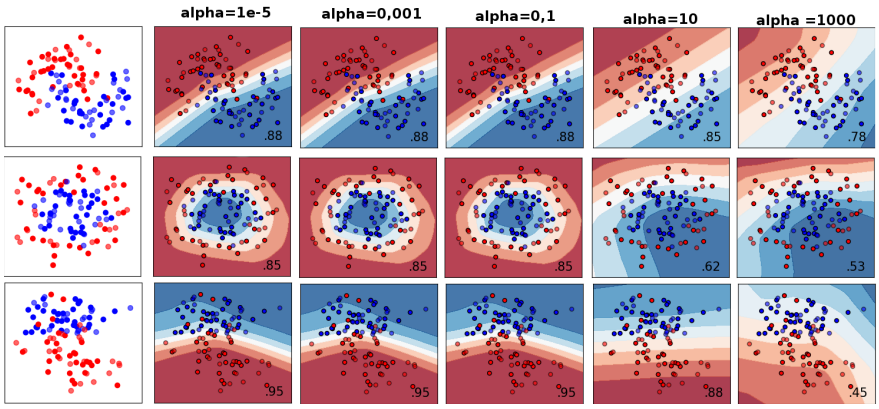


Figura 2.13: Hiperparametrización del valor alpha de la red neuronal.(Imagen extraída de la Web de Sklearn [17]).

Una de las hiperparametrizaciones más típicas es la del parámetro alpha (α), que es el encargado de cuantificar la penalización explicada en el apartado 2.7 para evitar el sobreajuste de la red. En la figura 2.13 podemos ver cómo se hiperparametriza dicho valor de forma esquemática en distintos problemas de clasificación y viendo de forma visual lo adecuado que es en cada caso.

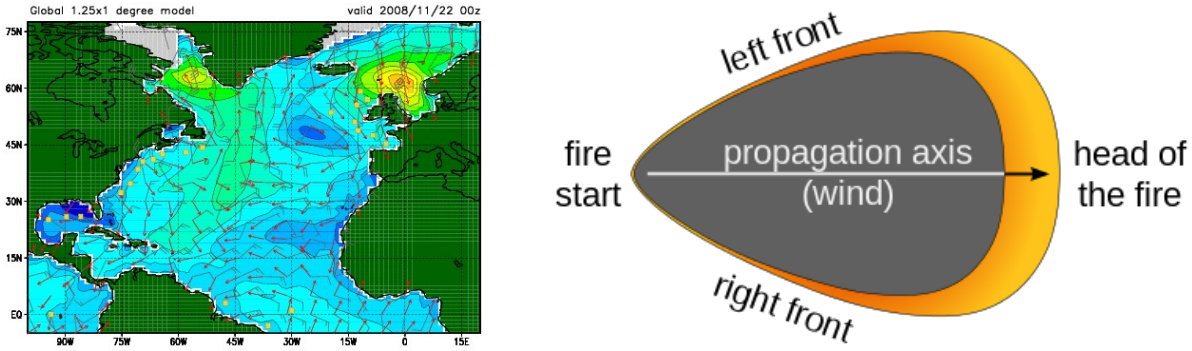
3

Predicción numérica del tiempo

3.1. Introducción

La predicción del tiempo atiende a modelos matemáticos de la mecánica de fluidos (aire en el caso de la atmósfera, agua en el caso de océanos) con el objetivo de predecir el estado futuro de los mismos. Esta dependencia matemática es bastante problemática por diferentes motivos. Entre otros problemas, dichas ecuaciones están formuladas de forma analítica, lo que es bastante complicado de tratar para un ordenador en términos de tiempo de computación.

Es aquí donde entra el modelo *Numerical Weather Prediction* (NWP) o predicción numérica del tiempo. Consigue eliminar el tratamiento de las ecuaciones en forma analítica de la mecánica de fluidos para tratarlo en forma de integración numérica teniendo como referencia estados anteriores del fluido para predecir estados posteriores. Es por esto que es necesario una inicialización de dicha integración numérica, para lo cual suele utilizarse el estado actual del fluido en cuestión.



(a) Modelo marino

(b) Modelo de propagación de un incendio

Figura 3.1: Ejemplos de modelado de la predicción del estado la superficie oceánica y del modelado de la predicción de propagación de un incendio que utilizan NWP . (Imágenes extraídas de Wikipedia [18]).

El primer acercamiento al modelo data de alrededor de la década de 1920 de la mano de Lewis Fry Richardson pero no fue hasta la década de 1950 que las especificaciones de los computadores (el

Electronic Numerical Integrator And Computer (ENIAC) más concretamente) fueron lo suficientemente buenas como para poder hacer una simulación de dicho modelo y que produjese resultados útiles.

Es necesario que los ordenadores sean suficientemente potentes para poder obtener unos resultados realmente interesantes ya que, por ejemplo, en la década de los años 20 se necesitaba de al menos seis semanas de trabajo para producir una predicción de seis horas desde el estado inicial, lo que resultaba poco interesante en términos de uso productivo.

3.1.1. Definición del modelo

El modelo se basa en dividir el fluido en cuestión en una malla que sea tratable para un sistema automatizado de forma sencilla, siendo un ejemplo de coordenadas la longitud, la latitud y la altura o la presión. En cada una de esas celdas es donde encontraremos las distintas predicciones del modelo para la región representada por cada celda.

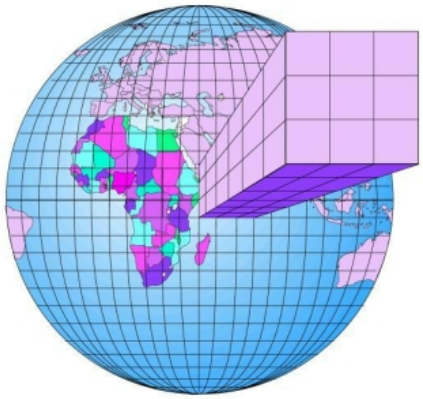


Figura 3.2: Imagen esquemática de la división tridimensional de la atmósfera en el modelo de NWP . (Imagen extraída de la Web de Neil Conway [19]).

Para establecer esa malla tridimensional es necesario escoger el sistema de coordenadas a utilizar. El sistema más común para las coordenadas horizontales suelen ser las coordenadas de latitud y longitud. Para establecer la coordenada vertical existen distintas opciones como la presión o la altura según los intereses particulares del modelo. En nuestro caso particular las coordenadas horizontales serán latitud y longitud y las diferencias verticales se establecerán como distintas componentes bajo la misma celda.

La atmósfera es un fluido y, como hemos visto, existen modelos para, dadas unas condiciones iniciales, predecir estados futuros del fluido en cuestión. El problema es que al establecer las condiciones iniciales, llevan un error asociado a las mismas (ya sea de medida directa de la componente introducida o de una predicción no completamente coincidente anterior), por lo que al tratar numéricamente esos valores, ese error se ve acrecentado a medida que utilizamos esos valores iniciales para operar.

"Model Output Statistics" (MOS)

Anteriormente hemos comentado que la predicción del estado de cualquier fluido lleva un error asociado, el cual se acrecenta a medida que nos distanciamos de la situación inicial presentada a la predicción. Es por esto que se hace necesario encontrar métodos de tratamiento de errores para los métodos de predicción numéricos con los que vamos a tratar.

Model Output Statistics (MOS) es un tipo de post-procesado estadístico de los datos basado en múltiples regresiones lineales con el objetivo de establecer relaciones estadísticas entre las variables que componen el sistema a estudiar con el fin de intentar corregir posibles errores predictivos de alguna de las componentes y así intentar ajustar más fielmente la predicción a los futuros datos reales y, como consecuencia, mejorar las siguientes predicciones.

3.2. La atmósfera como sistema caótico

Una de las complicaciones de predicción de la meteorología reside en que se trata de lo que se denomina un sistema caótico. Un sistema caótico o sistema complejo es aquel sistema que es extremadamente sensible a las pequeñas variaciones en las condiciones iniciales, lo que los hacen bastante difíciles de predecir a largo plazo. Uno de los sistemas caóticos más conocidos es el del "Atractor de Lorenz" presentado en el artículo *"Flujo determinista no periódico"* [20]. Este sistema nos interesa en gran medida ya que se trata de un sistema basado en ecuaciones simplificadas derivadas de las ecuaciones dinámicas de la atmósfera terrestre.

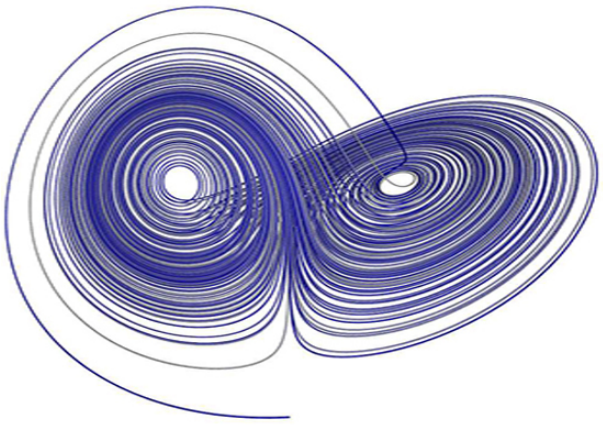


Figura 3.3: Atractor de Lorenz representado tridimensionalmente. (Imagen extraída de la Web "Axonométrica" [21]

Con el sistema representado en la figura 3.3, Edward Norton Lorenz trató de explicar el comportamiento caótico de algunos sistemas inestables, entre otros la predicción meteorológica. Así, introdujo el concepto extensamente conocido del "Efecto Mariposa", según el cual se preguntaba si "el simple aleteo de una mariposa en Brasil podría hacer aparecer un tornado en Texas", título de una conferencia

de la *American Association for the Advancement of Science* de la que él fue el autor [22].

Observando que la atmósfera atiende a un sistema tan delicado e inestable y dado que resulta bastante útil la predicción del estado de la misma, son necesarias técnicas para intentar hacer que la predicción sea más exacta. Una de estas técnicas es la predicción por conjuntos o "*ensemble forecasting*". Esta técnica consiste en realizar varias predicciones con el mismo modelo pero variando las condiciones iniciales mínimamente entre una ejecución y otra. Así se establece un conjunto de posibles futuros estados bajo los que puede acabar la atmósfera (en el caso de que ese sea el fluido de estudio). De esta forma se puede dar un abanico de posibilidades más grande e incluso intentar menguar los errores asociados con técnicas estadísticas como MOS vista en el apartado 3.1.1.

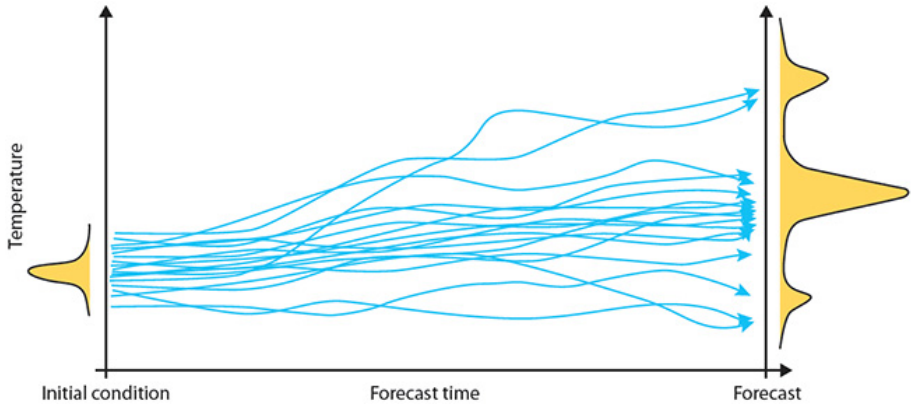


Figura 3.4: Imagen esquemática de la técnica de *"ensemble forecasting"* o predicción por conjuntos (Imagen extraída de la Web del ECMWF [23]).

En la figura 3.4 podemos observar esquemáticamente cómo se realizan una serie de predicciones con las condiciones iniciales ligeramente variadas y se es capaz de obtener varios estados futuros con distintas probabilidades según el número de predicciones realizadas y cuántas han terminado en estados similares.

3.3. Predicción del modelo

Pese a que existen una gran cantidad de modelos para predecir la meteorología, cada uno tiene sus características matemáticas y resultan más útiles para resolver un tipo de problemas que otros. En la figura 3.5 se puede ver la comparativa de precisión de la predicción según la distancia temporal de algunos de los modelos más importantes.

Utilizamos el modelado de predicción numérica del tiempo (NWP) debido a las especificaciones del problema a tratar, ya que siempre estaremos tratando con predicciones del mismo día en cuestión o como mucho a tres días vista y, a simple vista se ve que resulta el más óptimo para nuestro problema en particular.

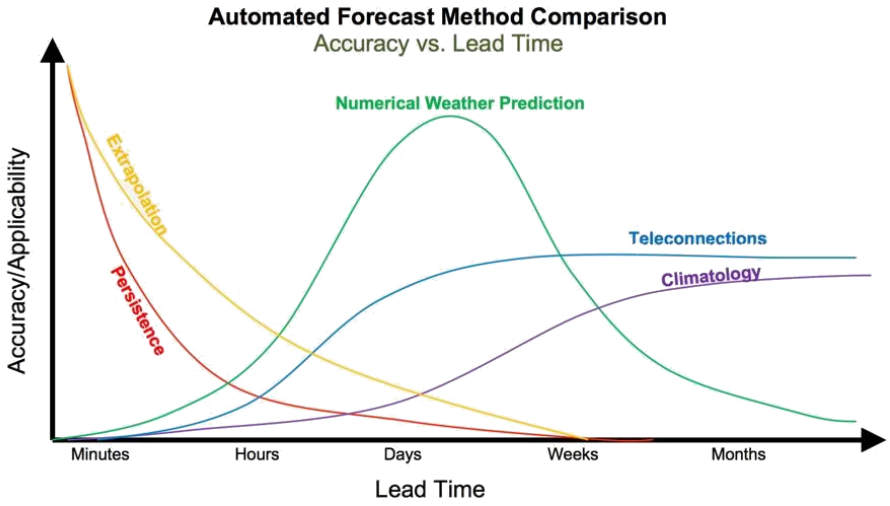


Figura 3.5: Comparación de precisiones de distintos modelos según la distancia de predicción. (Imagen extraída del Blog de Brent Shaw [24]).

En el concepto de la figura 3.5 se puede encontrar la motivación para éste trabajo, ya que tenemos que observar a qué distancia temporal se encuentran los datos de las predicciones más óptimos para resolver problemas de predicción.

3.4. *European Centre for Medium-Range Weather Forecasts*

El *European Centre for Medium-Range Weather Forecasts* (ECMWF) es un instituto de investigación y un servicio completamente operativo e independiente destinado a recabar datos y producir las predicciones numéricas del clima de las que hemos hablado anteriormente.

Se trata de un centro creado en el año 1975 por la cooperación europea de ciencia y tecnología (Web de COST [25]). El propósito principal por el que se fundó esta organización es proveer de predicciones atmosféricas de rango medio a los países europeos que lo conforman. En los últimos años ha crecido desde tener 22 países miembros (que son aquellos que participaron de la fundación de la organización) a añadir a 12 países cooperadores, conformando un total de 34 países cooperando para que el proyecto del ECMWF continúe adelante. Las principales misiones del ECMWF son:

- Producir predicciones numéricas del clima y monitorizar el sistema terreste.
- Investigar científica y técnicamente para que las predicciones lleguen a ser más precisas.
- Mantener un archivo histórico de datos meteorológicos.

Gracias a la colaboración internacional de los países que conforman la organización, cuenta con una gran cantidad de archivos de predicción numérica a una gran resolución. Así, tienen archivos

tanto en tiempo real como histórico de predicciones para usos tanto de explotación como meramente académicos. Entre otros, ofrecen predicciones numéricas del clima dos veces al día, análisis de la calidad del aire y de la circulación oceánica, y monitorización del clima y de la composición atmosférica. Para más información está la Web del ECMWF [26].

3.5. Archivos GRIB

3.5.1. Introducción a los archivos GRIB

Los archivos *GRIdded Binary* (GRIB) son archivos binarios comprimidos que contienen datos de meteorología. Estos archivos fueron creados por la *Organización Mundial de Meteorología* (OMM) con el fin de estandarizar y facilitar la transmisión de datos meteorológicos. Ha habido tres versiones de GRIB como se puede comprobar en la (Web sobre archivos GRIB): GRIB-0, GRIB-1 y GRIB-2.

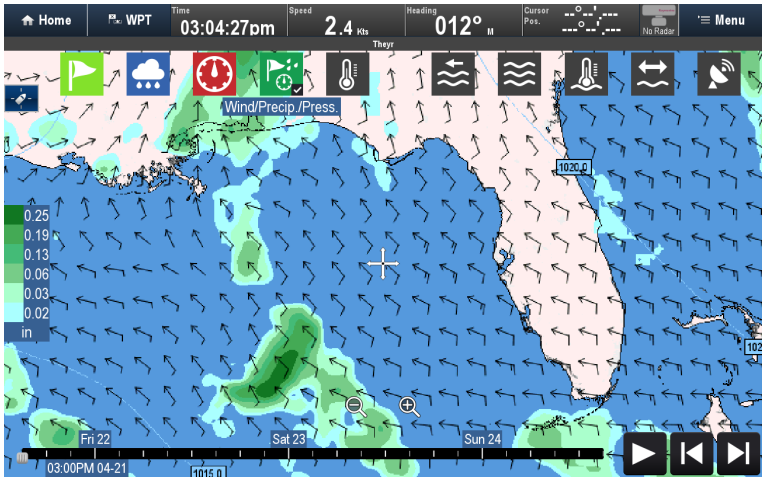


Figura 3.6: Visualización de los datos de un archivo GRIB a través del software *"GRIB Viewer"*. (Imagen extraída de la Web de Raymarine [27]).

GRIB-0 En desuso, obsoleto.

GRIB-1 Sigue estando reconocido por la OMM pero no se recomienda su uso ya que las organizaciones que producen los archivos GRIB no seguirán utilizando este formato y no se seguirá actualizando el software que trate con esta versión de archivo.

GRIB-2 Es el formato más moderno de los archivos GRIB y el más extendido y utilizado. Cabe destacar que no es retrocompatible con la versión anterior. En nuestro caso utilizaremos este tipo de archivo ya que será el que nos proporcionará el proveedor de dicha información.

3.5.2. Uso

Un archivo GRIB se descompone en una cadena de N mensajes, cada uno de los cuales tiene su cabecera y sus datos en binario. La cabecera contiene información como la calidad de los datos, meta-información de la cabecera en sí misma, métodos y parámetros utilizados para decodificar los datos en binario, etc.

Procesamiento en *Python*: la librería *Pygrib*

El procesamiento de este tipo de datos en *Python* es muy sencillo gracias a la librería *Pygrib*. Se trata igual que un archivo de texto solo que en vez de indicar los *offsets* en *bytes* a leer, se indica el número de mensajes a leer en la siguiente iteración. Aunque por otro lado una forma más simple de recorrer los mensajes de todo el archivo sería utilizando el *for* mejorado de *Python* sobre el objeto de apertura del fichero, con lo que iteraremos sobre la lista de mensajes.

Un mensaje es un objeto de la clase $GribMessage$. Un objeto $GribMessage$ actúa a modo de diccionario clásico añadiendo funcionalidades extra como la posibilidad de filtrar los datos de una zona concreta en base a las latitudes y longitudes que la delimitan (función $data$) o atributos tales como la fecha del análisis para la creación de los datos ($analDate$) o la fecha máxima de validez de los mismos ($validDate$). Así, para alcanzar los valores numéricos generados por el modelo que nos interesan utilizaremos la clave "$values$", lo que nos devolverá un $array$ de $numpy$ con la forma del grid en cuestión.

Un objeto de la clase $GribMessage$ tiene una serie de métodos que nos facilitan el manejo de los mismos como:

data(): Devuelve los valores de la región delimitada pasada por los parámetros $lat1$, $lat2$, $lon1$ y $lon2$. Así conseguimos generar un mensaje de tipo GRIB más ajustado a la zona seleccionada.

has_key(key): Devuelve un valor lógico comprobando si el valor pasado como parámetro se encuentra en el mensaje.

keys(): Devuelve la lista de valores que tiene el archivo GRIB en cuestión.

latlons(): Devuelve dos colecciones de números decimales representando la lista de longitudes y latitudes que abarca el mensaje a tratar.

Por otro lado tenemos los objetos de la clase $Index$, la cual nos permite explorar este tipo de archivos indexados por las claves pasadas como argumento en el constructor de la misma.

El uso de esta clase suele ser recomendable en los casos en los que se requiera de una gran eficiencia a la hora de filtrar mensajes pero no se recomienda en los casos en los que el fichero GRIB contenga campos "multivaluados" (con más de un valor).

4

EXPERIMENTACIÓN

4.1. Entorno de ejecución

4.1.1. Software

El desarrollo de las herramientas y $scripts$ a utilizar en este estudio será a través de $Python$. Más concretamente se realizarán en la versión de $Python\ 3$ debido a su gran extensión de librerías que serán necesarias para la realización del presente estudio. A través del entorno de $Miniconda$ (una versión reducida del entorno de $Anaconda$) instalaremos las siguientes librerías o módulos para su posterior utilización con el comando $conda\ install$.

Datetime Módulo de la librería nativa de $Python$ muy útil para el manejo de fechas que, combinada con $Pytz$ nos permite asegurar que tanto la separación a N días vista de la predicción meteorológica como la asociación de las producciones a las predicciones con independencia del huso horario pueden realizarse correcta y eficientemente.

Gzip Módulo de la librería nativa de $Python$ útil para la compresión y decompresión de archivos. Resulta realmente útil debido a la naturaleza tan pesada del estudio a realizar y de los grandes tamaños de archivos a tratar en el mismo. Así se permite una gran optimización en lo que al uso de disco se refiere a cambio de una pequeña carga computacional extra consistente en comprimir y descomprimir archivos.

Pandas Librería con la que podremos crear tablas indexadas (por fecha y hora en nuestro caso particular) para un manejo más sencillo y eficiente de la gran cantidad de datos de los que dispondremos. Para más información tenemos la Documentación de la librería Pandas

Pickle Módulo útil para la serialización de objetos $Python$ al que recurriremos habitualmente para poder realizar cada uno de los pasos del experimento de forma independiente.

Pygrib Esta librería nos simplificará el tratamiento de los archivos GRIB de predicción meteorológica de los que hemos hablado anteriormente, de tal forma que leer cada mensaje del archivo se reducirá a leer el archivo como si de un archivo de texto se tratara, intercambiando bytes de lectura por la consecución de mensajes que forman un archivo GRIB. Se

puede obtener información más en profundidad de la Web de Pygrib.

Matplotlib Librería necesaria para graficar los datos de los resultados obtenidos con el estudio para poder ver visualmente los resultados.

Sklearn Librería que proporciona la funcionalidad de los modelos predictivos que utilizaremos ($Ridge$ y $MLPRegressor$) así como métodos de procesamiento estadísticos para los errores obtenidos de los mismos (MAE y *Mean Squared Error* (MSE)) o métodos de preprocesamiento de datos ($StandardScaler$). Las especificaciones completas de éstos objetos se encuentran en la documentación de la librería Sklearn

4.1.2. Hardware

Para la ejecución de este estudio, el cual precisará de una gran capacidad computacional y de cálculo, fue habilitada una cuenta en el equipo ubicado en la ruta $eolo.iic.uam.es$ al cual nos conectaremos a través del comando de $bash$: ssh.

Dicha máquina pertenece al *Instituto de Ingeniería del Conocimiento* (IIC) en la cual se deja que este estudio se ejecute debido a su posible interés en los resultados del mismo dentro de su proyecto de predicción de energía eólica (EA2). Al tratarse de una máquina compartida y destinada a trabajos bastante cargados, la forma de ejecución entre usuarios se organiza a través de $slurm$, reservando espacio en la cola de ejecución con el comando $sbatch$, revisando la actual con $squeue$ y cancelando procesos activos con $scancel$. Las especificaciones técnicas de esta máquina son:

Procesador: 40 procesadores.

Cores: 10 núcleos por procesador.

Memoria RAM: 503 GigaBytes.

Memoria SWAP: 511 GigaBytes.

Sistema Operativo: Ubuntu 16.04.3 LTS

4.1.3. Obtención de datos

Lo más importante a la hora de comenzar este estudio se trata de encontrar unos datos de calidad de predicción numérica del clima correspondientes a la península ibérica. Dado que el IIC tenía esos datos para el proyecto EA2 anteriormente comentado, ya precisaban de esos archivos GRIB diarios de predicción meteorológica, los cuales compran al ECMWF . Asimismo, también disponían del archivo necesario de las producciones de energía eólica a nivel peninsular.

Ya hemos mencionado que los datos de predicción meteorológica vienen en formato GRIB y que vienen conformados como si fueran una malla bidimensional pero con componentes como para

establecer una tercera dimensión en cada una de las celdas. Dichos archivos tienen una resolución de $0{,}125^o \times 0{,}125^o$ en términos de latitud y longitud, lo que quizás resulte demasiada resolución sabiendo que se trata de una región tan grande como la península ibérica.

Por ello, en uno de los pasos del experimento dividiremos cada uno de los archivos GRIB presentados de forma diaria de tal forma que consigamos un subconjunto de esas celdas para obtener archivos de menor resolución como de $0{,}25^o \times 0{,}25^o$, de $0{,}5^o \times 0{,}5^o$ y de $1{,}0^o \times 1{,}0^o$. No se utilizarán para el experimento archivos de más resolución que los anteriormente nombrados por su excesiva carga computacional y por falta de datos anteriores a marzo del año 2015. Se hace necesaria la división de estas resoluciones para poder ir haciendo ajustes al modelo y no trabajar siempre con una gran cantidad de datos. De esta forma, se realizará el experimento en todas las resoluciones pero se presentarán los resultados con la resolución más precisa utilizable que, en este caso, se trata de la resolución $0{,}25^o \times 0{,}25^o$.

Los datos de producción eólica corresponden a la producción a nivel peninsular de la Red Electrica y vienen dados en un archivo tipo *Comma-Separated Values* (CSV) con las fechas de producción y los porcentajes de producción total resultantes de ese rango de tiempo. El tratamiento y complicaciones de este archivo se explicarán más extensamente más adelante. De forma alternativa, si no se dispusiese de este archivo de producciones se podrían obtener los datos de la Web de ESIOS.

4.2. Método experimental

En el presente estudio se tratarán los datos con diferentes $scripts$ y funciones escritos en el lenguaje $Python$ y, debido a la gran cantidad de datos manejados, cada uno de los resultados de cada paso será almacenado en un directorio, por lo que en la figura 4.1 se muestra la estructuración completa de la ejecución que se llevará a cabo. La documentación del código utilizado se encuentra en el anexo B.

Paso 1: Conversión a *Dataframes*

El primer paso del estudio consiste en coger los datos brutos del directorio en el que tenemos los archivos GRIB para pasarlos a un diccionario, a partir del cual será bastante sencillo pasar a un objeto $Dataframe$ de la librería $Pandas$. Cada uno de los archivos a convertir estarán indexados por la fecha de predicción y se le añadirá una columna extra de fecha de análisis, siendo ésta la fecha en la que fue generada dicha predicción.

Este será el paso en el que conseguiremos repartir los mismos datos en distintos archivos para poder escoger distintas resoluciones para el estudio, desechando aquellos datos cuya latitud o longitud no esté dentro de la resolución elegida por métodos de aritmética modular.

El código utilizado para tratar los archivos GRIB como diccionarios está basado en el código

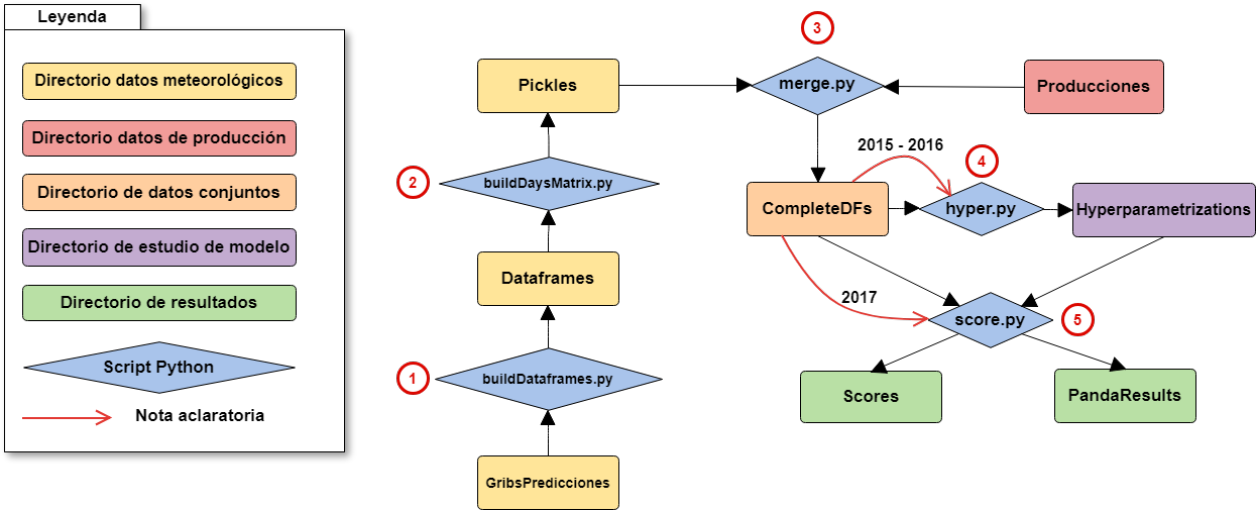


Figura 4.1: Diagrama de pasos de ejecución.

del *Trabajo de Fin de Grado* de Juan Bella *"Herramientas $Python$ para la predicción de energías renovables"* [28] añadiendo sobre él ligeras modificaciones para nuestro uso particular.

Paso 2: Creación de las matrices de datos a distintos días

En este caso, recorreremos todos los archivos generados en el paso anterior por resoluciones (a partir de éste momento supondremos que siempre se realiza el mismo tratamiento de los datos para cada una de las resoluciones con las que trabajaremos) y extraeremos aquellos datos que mantengan la diferencia de días que busquemos en cada caso. El código para la creación de las matrices de distancias esta también basado en el $Trabajo\ de\ Fin\ de\ Grado$ de Juan Bella anteriormente nombrado.

De esta forma conseguiremos la matriz de las predicciones que se generaron el mismo día, la matriz de las producciones que se generaron a un día vista y así sucesivamente. En nuestro caso realizaremos el estudio como máximo a una diferencia de dos días entre la fecha en la que se generó la predicción y para la que va destinada la misma ya que entendemos que la degradación de la predicción meteorológica es demasiado grande como para producir resultados interesantes a tres o más días.

Paso 3: Unión de producciones con predicciones

Este paso consiste en unir correctamente las producciones con el instante de tiempo al que corresponden. Para ello, lo primero que haremos será cargar el archivo CSV en el que están las producciones y convertirlo a otro objeto de tipo $Dataframe$ indexado también por la fecha a la que le corresponde cada producción. Cabe destacar que las fechas de las predicciones meteorológicas vienen en formato *Universal Time Coordinated* (UTC) , por lo que ya vienen en un huso horario que no depende de ninguna zona geográfica. Sin embargo, en nuestro caso particular de los datos de producción de energía eólica en la península ibérica sí que depende del huso horario de la misma, por lo que

tendremos que pasarlo a UTC previamente.

Una vez que tenemos los dos objetos de tipo $Dataframe$ indexados por las fechas correctas hacemos la intersección entre los dos conjuntos de datos ya que no nos interesa ninguna predicción meteorológica sin una producción asociada ni viceversa, ya que hemos escogido un aprendizaje supervisado para el desarrollo del estudio.

Tras este paso obtendremos ya los datos necesarios para establecer nuestras regresiones en distintos modelos, siendo todas las componentes meteorológicas pertinentes los valores de entrada a nuestra red neuronal en cada momento y la salida la predicción de producción de energía eólica. Para que la red neuronal sea capaz de tratar los datos de mejor forma será necesario hacer un preprocesamiento de los mismos. Ésto lo haremos a través del objeto de la librería $Sklearn$ llamado $StandardScaler$ que será el encargado de normalizar los datos para dicho fin.

Paso 4: Hiperparametrización

Este se trata del proceso más costoso computacionalmente del estudio. Gracias a los objetos de la librería $Sklearn$ llamados $GridSearchCV$ y $KFolds$, conseguiremos tanto establecer el espacio de búsqueda como el número de subconjuntos que haremos en la validación cruzada de cada uno de los modelos a probar en el espacio de búsqueda.

En el objeto $KFolds$ especificaremos que la validación cruzada de cada modelo será entrenando con tres cuartas partes de los datos y comprobando con el cuarto restante. Es importante especificar que no mezclaremos los datos de manera aleatoria ya que nos interesa que siempre tengamos cada época del año en el entrenamiento y al comprobar los resultados ya que la influencia de las estaciones anuales podría ser bastante determinante para los resultados.

Por otro lado el objeto $GridSearchCV$ nos permite establecer el espacio de búsqueda del número de hiperparámetros que queramos para intentar buscar el modelo más óptimo posible para la solución de nuestro problema variando los hiperparámetros que le especifiquemos. Como no es la intención de este estudio encontrar el mejor modelo posible, sólo hiperparametrizaremos el parámetro que indica la cantidad de penalización que estableceremos al ajuste de la curva al conjunto de datos presentados, es decir, el parámetro $alpha$ encargado de la regularización de la red neuronal 2.7.

Como podemos ver en el diagrama de ejecución del estudio 4.1 sólo utilizaremos los datos de 2015 y 2016 para la hiperparametrización ya que no nos interesaría que el conjunto de test elegido (2017) contaminase la elección de los parámetros de la red neuronal o el entrenamiento de la misma pues si no, no podremos simular un estado de explotación de la red y no la estaríamos evaluando correctamente.

A través de esta estrategia conseguiremos ver la curva de hiperparametrizaciones para cada configuración de horizontes de los modelos y así determinar qué parámetro $alpha$ resulta más óptimo para

nuestro problema y cómo de crítica resultaría su elección arbitraria.

Paso 5: Obtención de resultados

A partir de los datos completos y de los datos obtenidos de la hiperparametrización se entrenarán una serie de modelos con los valores óptimos obtenidos de la hiperparametrización con los años 2015 y 2016, para posicionar el año 2017 como año de test como hemos comentado anteriormente. Entrenar varios modelos (en nuestro caso haremos cinco) y promediar los resultados de los mismos nos servirá para minimizar el impacto que haya podido tener un entrenamiento que haya podido llegar a un mínimo local.

Por último, éste paso tendrá dos salidas. Una de las salidas será una gráfica con las producciones reales frente a las predichas de tal forma que se pueda observar cuánto se aleja o acerca la tendencia general de predicción de la red neuronal y, por otro lado, un $Dataframe$ en el que se puedan observar las medidas estadísticas de distintos modelos expuestos a distintos tests. En nuestro caso particular serán los tres modelos entrenados a 0 días vista, a 1 día vista y a 2 días vista frente a tests de predicciones generadas a su misma vez a 0 días vista, a 1 día vista y a 2 días vista. De esta forma encontraremos 9 resultados estadísticos para cada una de las configuraciones propuestas para el estudio.

4.3. Primer acercamiento al problema: Modelado *Ridge*

En este primer acercamiento a la resolución del problema utilizaremos un modelado $Ridge$. Un modelado $Ridge$ nos permite establecer una regresión lineal simple entre los datos de entrada y el de salida para comprobar que distintos pasos del método experimental anteriormente expuesto son correctos, y supone una primera suposición de los resultados a obtener a la par que un punto de partida que mejorar en futuros modelados.

Este tipo de modelado se realiza de forma sencilla y rápida ya que requiere de muchos menos parámetros que escoger pues se trata de un modelo monocapa (es decir, no tiene capas ocultas), por lo que la cantidad de pesos a actualizar es radicalmente menor.

Por último, escogemos modelar con un modelo tan sencillo ya que la curva de hiperparametrización podrá verse de forma más clara y tendremos algún tipo de comportamiento que esperar para modelos posteriores.

Hiperparametrización

Para este proceso de hiperparametrización serán necesarios los objetos $GridSearchCV$ y $KFolds$ (como se ha mencionado anteriormente en el apartado 4.2) para recorrer el espacio de búsqueda de

hiperparámetros y realizar una validación cruzada para cada una de las combinaciones respectivamente.

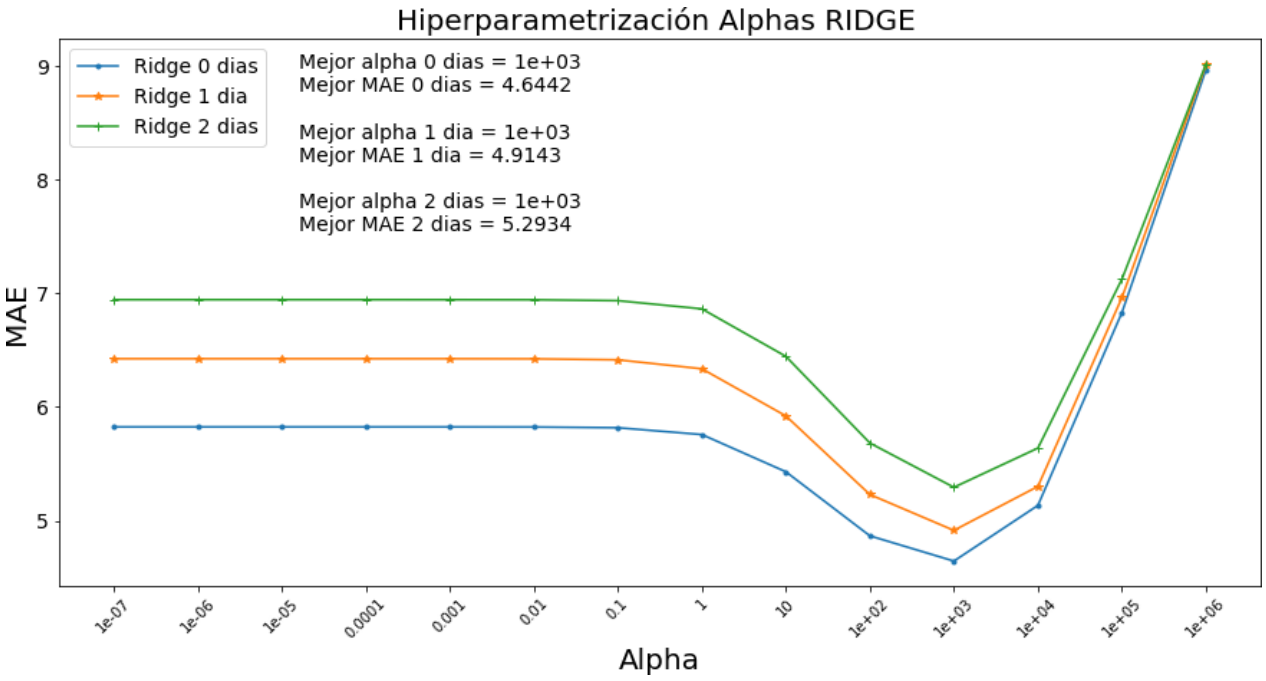


Figura 4.2: Hiperparametrización a distintos días vista de los modelos Ridge a $0{,}25^o$ de resolución.

Como podemos observar en la figura 4.2 se ha hiperparametrizado el parámetro $alpha$ de tres modelos de tipo $Ridge$ distintos: entrenado a cero días vista de la predicción, a un día vista y a dos días vista. Los valores de $alpha$ escogidos han sido las potencias de diez desde 10^{-7} hasta 10^6 y se han utilizado 4 $folds$ para validar cada uno de los valores correctamente. En la figura se puede observar como resulta claramente más óptimo uno de los valores que el resto ($\alpha = 10^3$) y que la predicción es más precisa cuanto más recientes sean los valores meteorológicos.

Resultados de estudio temporal

Para evaluar correctamente cada modelo en cuestión, lo que haremos será entrenarlo con el valor de $alpha$ óptimo con los años 2015 y 2016 para establecer el año 2017 como conjunto de test. En la tabla 4.1 se puede ver como a cada uno de los modelos les sometemos a pruebas de predicciones hechas a distintos días de diferencia entre la predicción y la fecha a la que está destinada la misma ($TestDay$ en la tabla).

TestDay	RIDGE-0	RIDGE-1	RIDGE-2
0	**MAE:** 4.49198 **STD:** 3.86874	**MAE:** 4.51995 **STD:** 3.79995	**MAE:** 4.68292 **STD:** 3.99597
1	**MAE:** 4.61794 **STD:** 3.96317	**MAE:** 4.63746 **STD:** 3.88824	**MAE:** 4.69138 **STD:** 4.13597
2	**MAE:** 4.80605 **STD:** 4.11206	**MAE:** 4.75215 **STD:** 4.05565	**MAE:** 4.95340 **STD:** 4.23195

Tabla 4.1: Tabla de resultados de estudio de los modelos Ridge a $0{,}25^o$ de resolución.

También se puede observar que, en la misma tabla, se han resaltado los mejores modelos para cada uno de los distintos días de $Test$ elegidos, de tal forma que en el caso de que quisiéramos predecir producciones para hoy o para mañana con el modelo $Ridge$ deberíamos escoger el modelo entrenado con predicciones hechas el mismo día, mientras que si buscásemos la mejor opción para pasado mañana utilizaríamos el modelo entrenado con valores a un día de diferencia.

Calidad de la predicción

La idea subyacente a este estudio es poder hacer la mejor predicción sea cual sea la combinación del modelo y la predicción que haya que utilizar ya que sería la más relevante a la hora de llevar este proceso a una producción real.

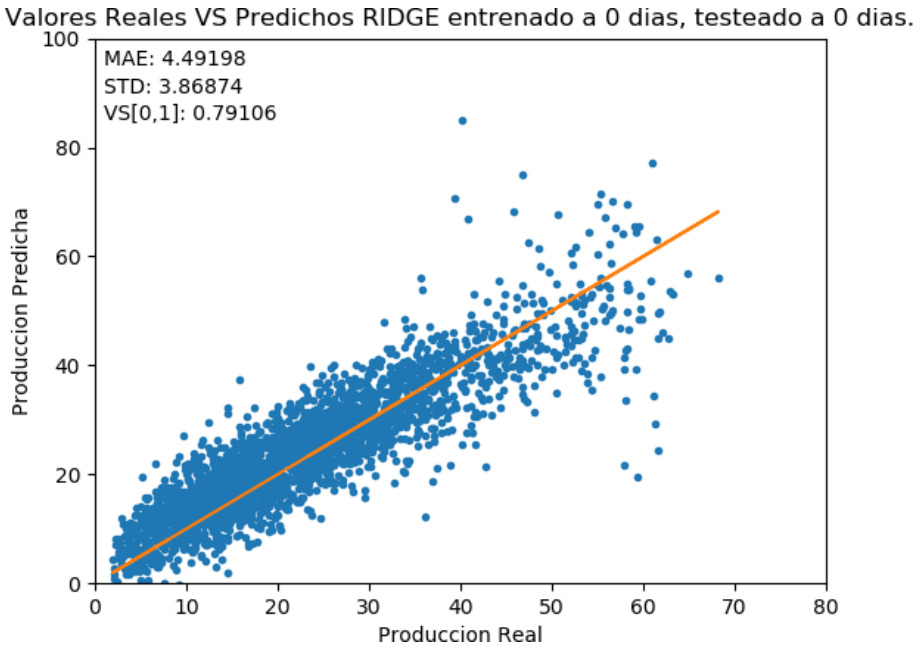


Figura 4.3: Dispersión de valores en un modelo Ridge a $0{,}25^o$ de resolución entrenado a 0 días vista y testeado a 0 días vista.

En la figura 4.3 se muestra una gráfica con la mejor combinación obtenida en la tabla 4.1 sobre las predicciones reales frente a las obtenidas por el modelo, la cual corresponde al modelo entrenado y testeado con los datos más recientes de predicción meteorológica. La recta $f(x) = x$ representaría la predicción perfecta, por lo que puntos más alejados estarían más equivocados que puntos cercanos.

Podemos ver cómo existe una cierta dispersión de los valores frente a la recta expuesta que se va acrecentando según avanzamos a producciones mayores. Suponemos que esto se debe a la impredictibilidad de los parques eólicos frente a condiciones extremas de viento ya que es posible, por ejemplo, que se decida cambiar el ángulo de las palas del molino según decisiones humanas con una cierta aleatoriedad.

4.4. Estudio con perceptrón multicapa con 2 capas ocultas de 100 neuronas.

Hiperparametrización

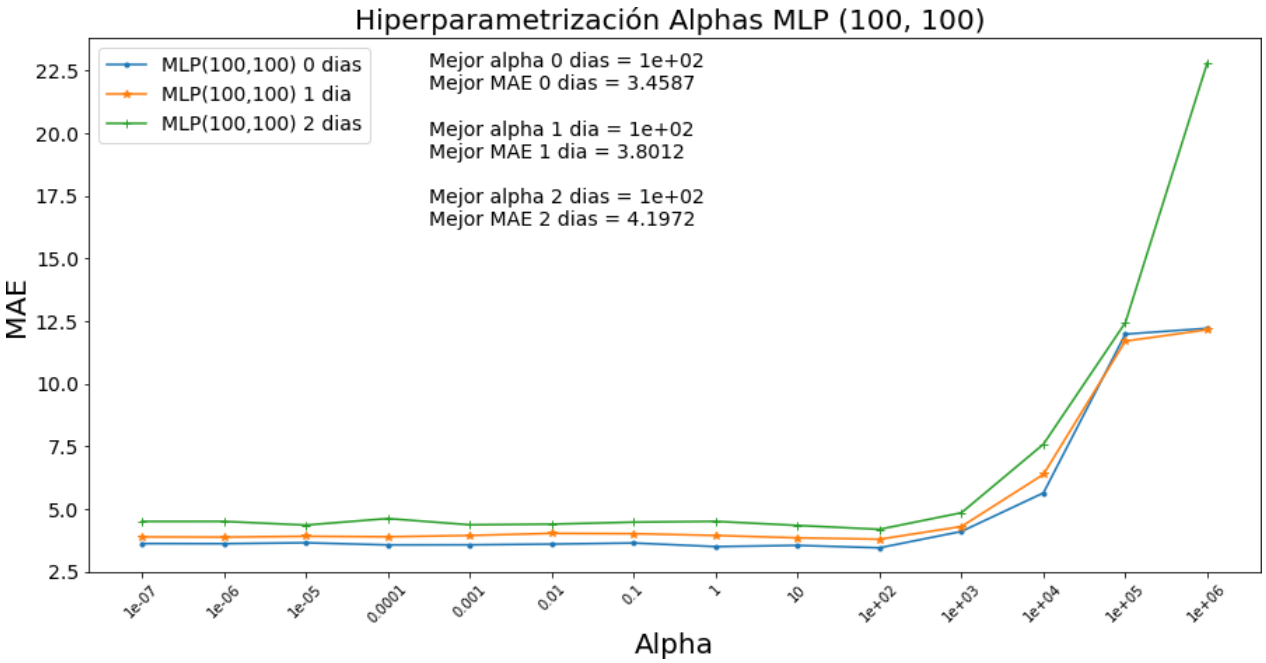


Figura 4.4: Hiperparametrización a distintos días vista de los modelos MLP con dos capas ocultas de 100 neuronas de a $0{,}25^o$ de resolución.

En la figura 4.4 podemos ver el proceso de hiperparametrización de los modelos entrenados con datos a distintos días de diferencia desde la fecha de la predicción de un *MultiLayer Perceptron* (MLP) con una topología de dos capas ocultas con cien neuronas por capa.

A diferencia con el modelo $Ridge$ anteriormente presentado en el apartado 4.3, se puede observar que no existe una gran diferencia de rendimiento a valores de $alpha$ pequeños, mientras que el error se dispara según lo vamos acrecentando. También podemos observar que el valor óptimo de $alpha$ para cada uno de los modelos es $\alpha = 10^2$ y los mejores MAEs obtenidos son $3{,}4587$, $3{,}8012$ y $4{,}1972$ para los modelos a cero, uno y dos días respectivamente.

También se observa que cuanto más recientes resultan los datos de la predicción, más exactos son los resultados para la prueba de cada valor de $alpha$, lo que concuerda con lo que se espera *a priori* como resultado de este estudio, que los valores de predicción meteorológicas más recientes sean más eficientes a la hora de predecir la producción de energía eólica.

Resultados de estudio temporal

Rompiendo con los resultados vistos anteriormente tanto en la tabla de los resultados de $Ridge$ 4.1 y con la tendencia a ofrecer mejores resultados a predicciones más recientes explicada en el proceso de hiperparametrización de esta misma topología en el apartado 4.4, en la tabla de resultados 4.2 podemos ver cómo resultan más óptimos los modelos entrenados con datos a dos días vista en el caso de querer predecir la producción del mismo día o a un día vista, mientras que en el caso de querer predecir la producción de pasado mañana, resulta mejor el modelo entrenado con datos del mismo día.

TestDay	MLP-0	MLP-1	MLP-2
0	**MAE:** 3.26053 **STD:** 2.99888	**MAE:** 3.54674 **STD:** 3.33629	**MAE:** 3.21176 **STD:** 2.92149
1	**MAE:** 3.63466 **STD:** 3.44785	**MAE:** 3.63604 **STD:** 3.32443	**MAE:** 3.45562 **STD:** 3.21304
2	**MAE:** 3.43038 **STD:** 2.91105	**MAE:** 3.88897 **STD:** 3.38762	**MAE:** 3.81415 **STD:** 3.22389

Tabla 4.2: Tabla de resultados de estudio de los modelos MLP de 2 capas de 100 neuronas a $0{,}25^o$ de resolución.

Calidad de la predicción

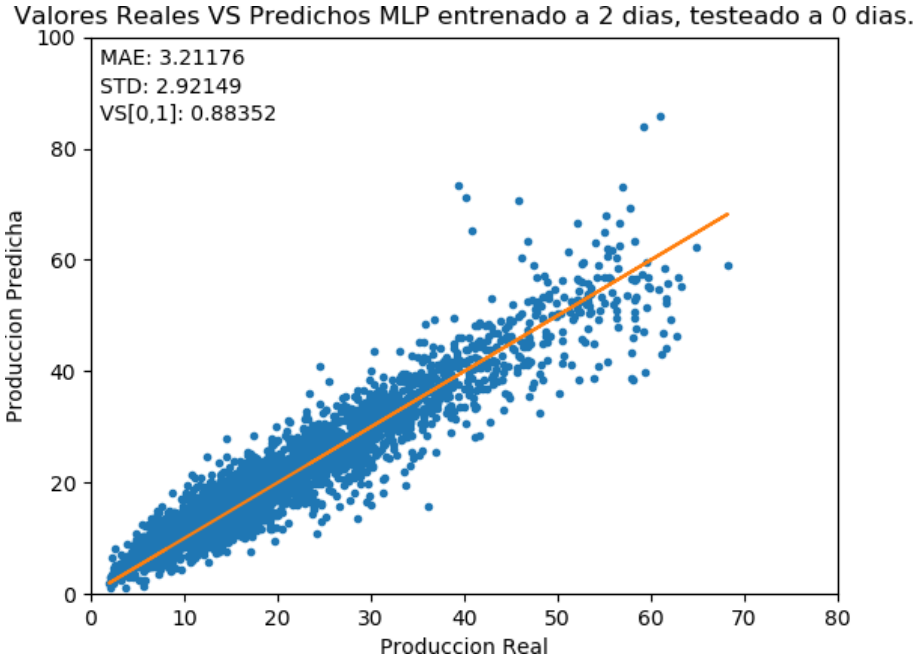


Figura 4.5: Dispersión de valores en un modelo MLP con dos capas ocultas de cien neuronas cada una a $0{,}25^o$ de resolución entrenado a 2 días vista y testeado a 0 días vista.

En la figura 4.5 podemos ver cómo la dispersión de los puntos es menor que la que se puede ver en la figura de la calidad de la predicción del modelo $Ridge$ 4.3, siendo los valores estadísticos de error medio, desviación típica y "$variance\ score$" más óptimos en éste caso, así como visualmente

se pueden ver valores menos dispersos de la recta $f(x) = x$. También podemos observar el mismo efecto descrito en la calidad de la predicción del modelo $Ridge$ 4.3, que los valores de producciones más altas son más dispersos que en el resto de casos.

4.5. Estudio con perceptrón multicapa con 2 capas ocultas de 200 neuronas.

Hiperparametrización

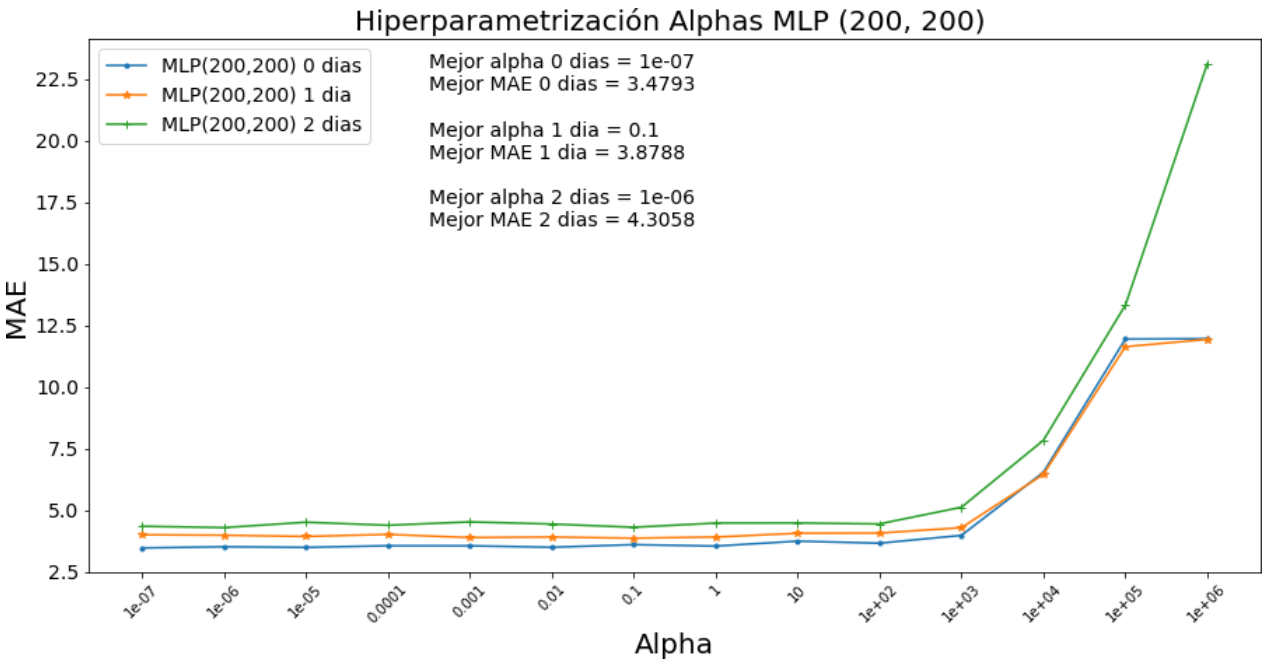


Figura 4.6: Hiperparametrización a distintos días vista de los modelos MLP con dos capas ocultas de 200 neuronas de a $0{,}25^o$ de resolución.

En la figura 4.6 podemos ver el proceso de hiperparametrización de los modelos entrenados con datos a distintos días de diferencia desde la fecha de la predicción de un MLP con una topología de dos capas ocultas con doscientas neuronas por capa.

Podemos ver que en este caso los valores de $alpha$ no son unánimes como en los casos de las hiperparametrizaciones de las topologías anteriores, sino que son $\alpha = 10^{-7}$, $\alpha = 0{,}1$ y $\alpha = 10^{-6}$ para los modelos de cero días vista, un día vista y dos días vista respectivamente, al igual que los valores óptimos de errores serán $3{,}4793$, $3{,}8708$ y $4{,}3058$ respetando el orden anterior. De nuevo se cumple que en el proceso de hiperparametrización las predicciones dan resultados mejores con las predicciones más recientes.

Resultados de estudio temporal

Continuando en la línea de los resultados de los modelos MLP con dos capas de cien neuronas por capa 4.2, en la tabla 4.3 que representa los resultados de esta nueva arquitectura de red volvemos a comprobar que no son los valores de predicciones más recientes los que resultan más eficientes a la hora de predecir la producción eólica. En este caso, para hacer la mejor predicción del mismo día en el que se produce la predicción meteorológica habría que utilizar el modelo entrenado a un día vista, mientras que en el resto de supuestos sería más óptimo el modelo entrenado con predicciones a dos días vista.

TestDay	MLP-0	MLP-1	MLP-2
0	**MAE:** 3.37294 **STD:** 3.16387	**MAE:** 3.21869 **STD:** 3.00399	**MAE:** 3.27078 **STD:** 3.11451
1	**MAE:** 3.60070 **STD:** 3.25950	**MAE:** 3.60712 **STD:** 3.34713	**MAE:** 3.44309 **STD:** 3.23667
2	**MAE:** 3.67842 **STD:** 3.28576	**MAE:** 4.00133 **STD:** 3.63607	**MAE:** 3.58069 **STD:** 3.15973

Tabla 4.3: Tabla de resultados de estudio de los modelos MLP de 2 capas de 200 neuronas a $0{,}25^o$ de resolución.

Calidad de la predicción

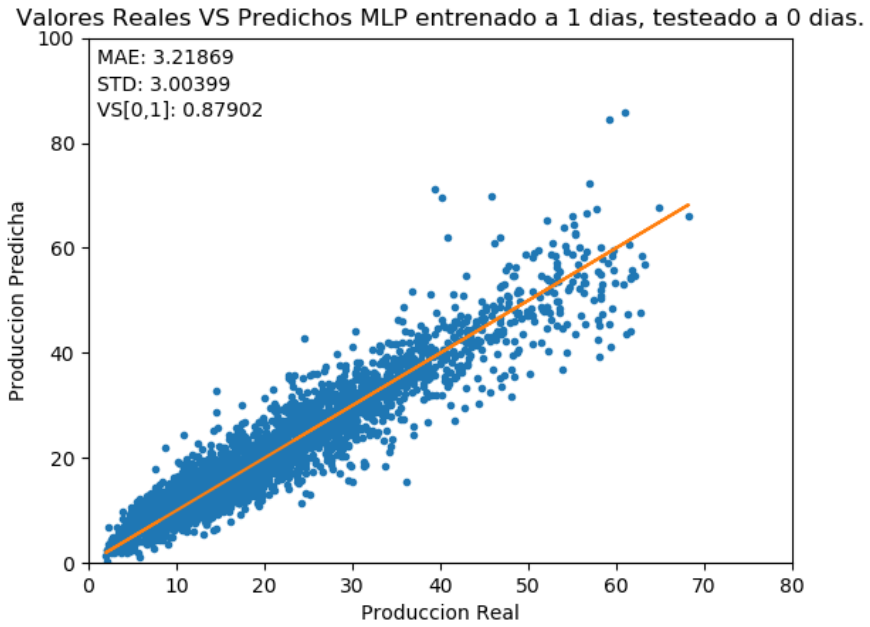


Figura 4.7: Dispersión de valores en un modelo MLP con dos capas ocultas de doscientas neuronas cada una a $0{,}25^o$ de resolución entrenado a 1 días vista y testeado a 0 días vista.

Como podemos observar en la figura 4.7, también se consigue mejorar los resultados estadísticos del modelado básico de $Ridge$ 4.3, pero aparecen ligeramente menos óptimos que la topología anterior de dos capas ocultas con cien neuronas cada una del apartado 4.4.

4.6. Resumen de resultados

En una primera impresión del problema cabría esperar que fuesen los modelos entrenados con datos más recientes los que diesen mejores resultados a la hora de predecir la producción eólica a distintos horizontes diarios. Esta intuición se iba haciendo más sólida según se veían los resultados de las curvas de hiperparametrizaciones ya que en ellas se observa una tendencia en esta dirección.

Sin embargo, a la hora de someter a los modelos entrenados a una fase de *Test* con un conjunto distinto de datos para simular una fase de explotación los resultados varían a lo esperado. No se puede ver una tendencia clara como para afirmar qué estrategia sería la más óptima para resolver el problema. Se puede ver que las desviaciones típicas de los errores cometidos en la predicción son de una magnitud similar al error medio cometido, por lo que tampoco se puede afirmar que los modelos ganadores lo sean en realidad y no se trate de un simple evento estadístico. Así se puede afirmar que es un campo que merece la pena seguir estudiando para conseguir unos resultados concluyentes ya que la hipótesis propuesta no se ha cumplido a pesar de parecer la más racional. Se pueden acreditar estos resultados en las comparaciones establecidas en el anexo A.

También se ha comprobado que no siempre las topologías más complejas tienen por qué superar en rendimiento final a las más sencillas, ya que por ejemplo en nuestro caso particular un MLP de dos capas de cien neuronas cada una supera en rendimiento a una arquitectura de red más compleja como es el caso del MLP de dos capas ocultas con doscientas neuronas por capa.

5

Conclusiones y trabajo futuro

5.1. Conclusiones

Este estudio ha servido como introducción a la utilización de herramientas de $machine\ learning$ y más en concreto de $deep\ learning$ así como para establecer un acercamiento a la matemática que encierran los modelos utilizados y medidas comparativas estadísticas entre los mismos. También se han adquirido destrezas en el manejo de archivos de datos voluminosos de predicciones numéricas y habilidades para tratarlos a través de tablas indexadas, así como la comprensión y el manejo de un modelado de predicción meteorológica como es la predicción numérica del tiempo o "$Numerical\ Weather\ Prediction$" para su uso en predicción de la producción eólica.

El campo del $machine\ learning$ es un campo en auge por su gran aplicabilidad en la resolución de problemas en todo tipo de ámbitos así como por su interés meramente científico debido a su gran capacidad para el procesamiento de grandes datos y el establecimiento de correlaciones entre variables para problemas tanto de predicción de valores como de clasificación en distintas clases. Adquirir destreza en este campo de conocimiento resulta realmente útil en un mundo cada vez más orientado a utilizar este tipo de técnicas para optimizar y automatizar cada proceso que se preste a hacerlo.

El objetivo de este *Trabajo de Fin de Grado* consiste en estudiar qué predicciones resultan más óptimas para cada horizonte de distancia temporal a la hora de predecir la producción eólica de una determinada región (en éste caso de la Red Eléctrica de España a nivel peninsular) a través de técnicas de $deep\ learning$. A la vista de los resultados obtenidos, se puede afirmar que la hipótesis que se quería comprobar no se cumpliría en sentido estricto. Ésto abriría una posible línea de investigación sobre el tema. Se puede afirmar que merece la pena estudiarlo por las implicaciones que tiene pues con una gran precisión en la predicción de producción eólica (u otra energía de carácter renovable) combinada con una buena predicción de la demanda energética, nos permitiría prescindir de las energías no renovables en los casos en los que la demanda no supere la producción de energías renovables y así contribuir a un sistema energético sostenible.

5.2. Trabajo futuro

El resultado obtenido abriría nuevas preguntas a responder. Sería interesante continuar con el experimento ampliando los días tanto de entrenamiento como de test, así como la prueba de nuevas topologías o estrategias de predicción. Podría estudiarse una red convolucional al tratar con datos situados espacialmente o también una red recurrente para intentar menguar el error en aquellos casos que, por injerencia humana no predecible, se ha modificado el rendimiento de la producción eólica. Para éste último caso además se precisaría de una comparativa estadística más rigurosa.

Otra línea de investigación que surgiría sería continuar la investigación con una zona más reducida y específica. Concretamente el parque eólico de Sotavento de Galicia también colabora con el proyecto del IIC encargado de esta área, el proyecto EA2.

Como ya he dicho, este *Trabajo de Fin de Grado* suscita más preguntas que respuestas, pero no hay que menospreciar este hecho ya que parafraseando al filósofo vienés Karl Raimund Popper y recalcando la cita que abre el presente documento: *"La ciencia será siempre una búsqueda, jamás un descubrimiento real. Es un viaje, nunca una llegada."*

Bibliografía

[1] R. Sabbatini, "Neurons and synapses: the history of its discovery." http://www.cerebromente.org.br/n17/history/neurons3_i.htm, 2003.

[2] "Wikipedia | Neurona." https://es.wikipedia.org/wiki/Neurona.

[3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, 1943.

[4] F. Sancho Caparrini, "Redes Neuronales: una visión superficial." http://www.cs.us.es/~fsancho/?e=72.

[5] J. Schmidhuber, "Deep Learning in neural networks: An overview," 2015.

[6] A. Dertat, "Applied Deep Learning.." https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6.

[7] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI International Joint Conference on Artificial Intelligence*, 2011.

[8] "New supercomputer on a chip 'sees' well enough to drive a car someday | Kurzweil." http://www.kurzweilai.net/new-supercomputer-on-a-chip-'sees'-well-enough-to-drive-a-car-someday.

[9] "Recurrent vs Hopfield neural networks - Cross Validated." https://stats.stackexchange.com/questions/99967/recurrent-vs-hopfield-neural-networks.

[10] D. E. Ruineihart, G. E. Hint, and R. J. Williams, "LEARNING INTERNAL REPRESENTATIONS BERROR PROPAGATION," 1985.

[11] F. Mallea, "Introducción a Machine Learning: Algunos conceptos básicos – Llipe." http://www.llipe.com/2017/03/13/introduccion-a-machine-learning/.

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2014.

[13] Y. Yue, "What is high bias and high variance in machine learning terminology in simplest terms?." https://www.quora.com/What-is-high-bias-and-high-variance-in-machine-learning-terminology-in-simplest-terms.

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks,"

[15] D. Stutz, "Understanding the difficulty of training deep feedforward neural networks explanation." http://davidstutz.de/understanding-the-difficulty-of-training-deep-feedfoward-neural-networks/.

[16] "Wikipedia | Validación cruzada." https://es.wikipedia.org/wiki/Validacion_cruzada.

[17] "Varying regularization in Multi-layer Perceptron — scikit-learn 0.19.1 documentation." http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html.

[18] "Wikipedia | NWP." https://en.wikipedia.org/wiki/Numerical_weather_prediction.

[19] N. Conway, "Numerical Weather Prediction." http://www.neilconway.org/talks/weather/.

[20] E. N. Lorenz, “Deterministic Nonperiodic Flow,” *Journal of the Atmospheric Sciences*, 1963.

[21] “Sobre la Teoría del Caos | axonométrica.” https://axonometrica.wordpress.com/2013/09/02/sobre-la-teoria-de-caos-2/.

[22] E. N. Lorenz, “Predictability; Does de Flap of a Butterfly’s wings in Brazil Set Off a Tornado in Texas?,” (Cambridge), 1972.

[23] “Twenty-five years of ensemble forecasting | ECMWF.” https://www.ecmwf.int/en/about/media-centre/news/2017/twenty-five-years-ensemble-forecasting.

[24] B. Shaw, “Weather Forecasting: How Does It Work, and How Reliable Is It?.” http://www.precisionag.com/systems-management/data/weather-forecasting-how-does-it-work-and-how-reliable-is-it/.

[25] “COST | Home.” http://www.cost.eu/.

[26] “What we do | ECMWF.” https://www.ecmwf.int/en/about/what-we-do.

[27] “Aplicación meteorológica GRIB Viewer.” http://www.raymarine.es/view/?id=17059.

[28] J. Bella Santos and J. R. Dorronsoro Ibero, “HERRAMIENTAS PYTHON PARA LA PREDICCIÓN DE ENERGÍAS RENOVABLES,” 2018.

Acrónimos

AdaGrad *Adaptative Gradient Algorithm*.

ADAM *ADAptative Moment estimation*.

CSV *Comma-Separated Values*.

ECM *Error Cuadrático Medio*.

ECMWF *European Centre for Medium-Range Weather Forecasts*.

ENIAC *Electronic Numerical Integrator And Computer*.

GRIB *GRIdded Binary*.

IIC *Instituto de Ingeniería del Conocimiento*.

LBFGS *Limited-memory BFGS*.

MAE *Mean absolute Error*.

MLP *MultiLayer Perceptron*.

MOS *Model Output Statistics*.

MSE *Mean Squared Error*.

NWP *Numerical Weather Prediction*.

OMM *Organización Mundial de Meteorología*.

RMSE *Root Mean Squared Error*.

RMSProp *Root Mean Square Propagation*.

SGD *Stochastic Gradient Descent*.

UTC *Universal Time Coordinated*.

Anexos

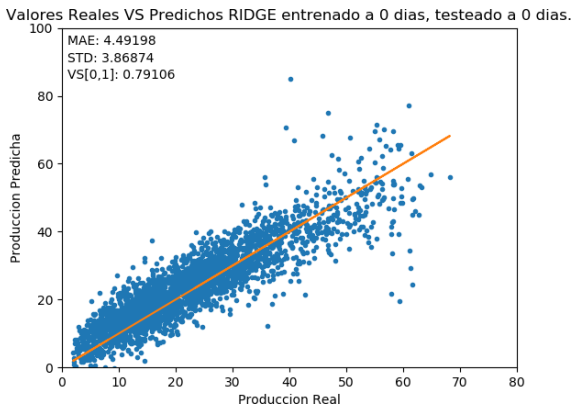
A

Comparativas de la calidad de la predicción de los distintos modelos.

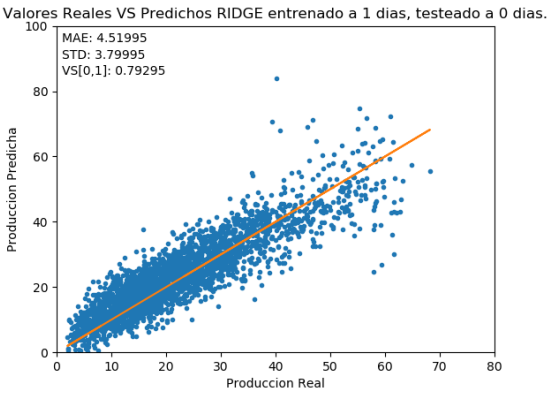
A continuación se mostrarán las gráficas de predicciones comparativas entre la predicción del modelo entrenado con los datos más reciente junto al ganador en cada caso, y, en los casos en el que el modelo más reciente fuera el ganador la comparativa se realizará con el segundo modelo más óptimo.

Como se puede comprobar, a pesar de que los valores estadísticos varían, no se pueden apreciar grandes diferencias cualitativas en las comparaciones de los modelos.

A.1. Modelo Ridge

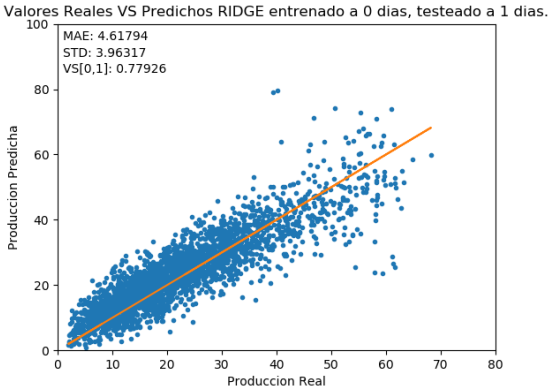


(a) Modelo más eficiente

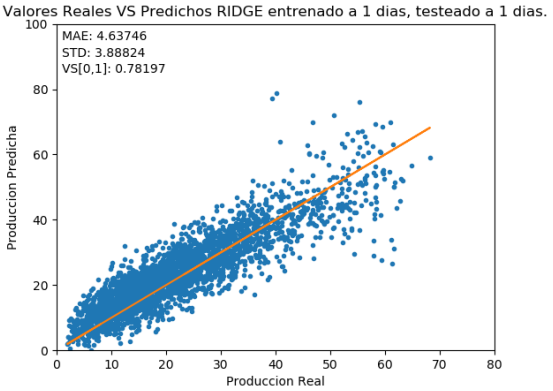


(b) Segundo modelo más eficiente

Figura A.1: Comparativa de test para horizonte de 0 días en el modelo Ridge.

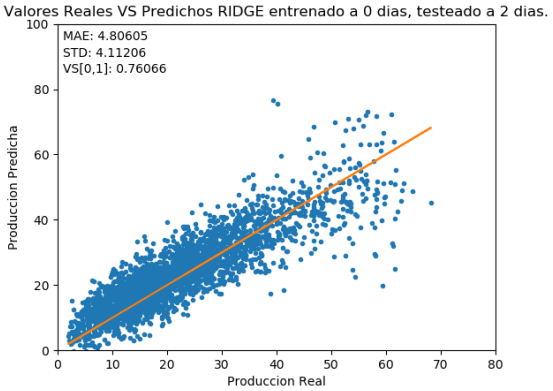


(a) Modelo más eficiente

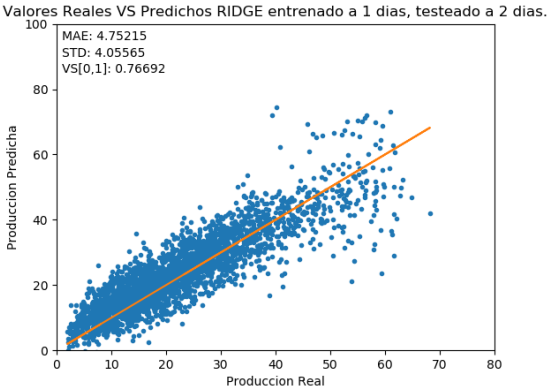


(b) Segundo modelo más eficiente

Figura A.2: Comparativa de test para horizonte de 1 día en el modelo Ridge.



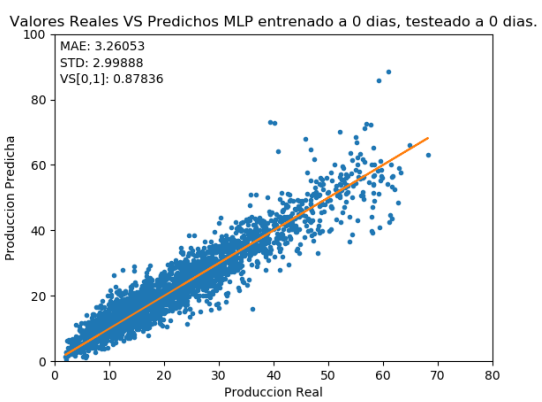
(a) Modelo más reciente



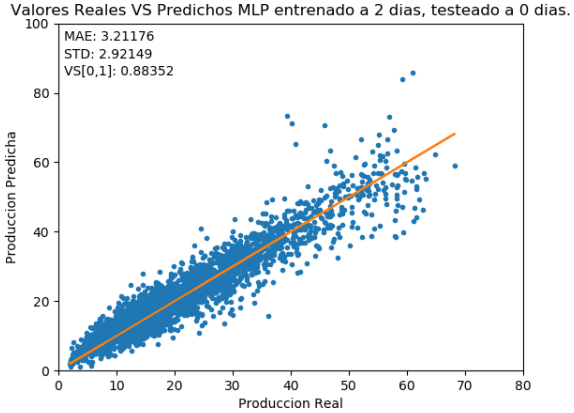
(b) Modelo más eficiente

Figura A.3: Comparativa de test para horizonte de 2 días en el modelo Ridge.

A.2. Modelo *MLP* con dos capas ocultas de cien neuronas cada una.

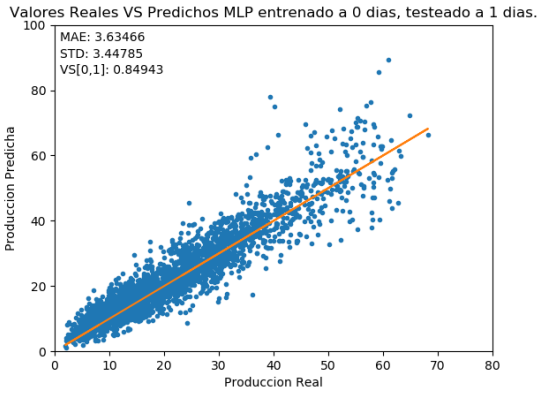


(a) Modelo más reciente

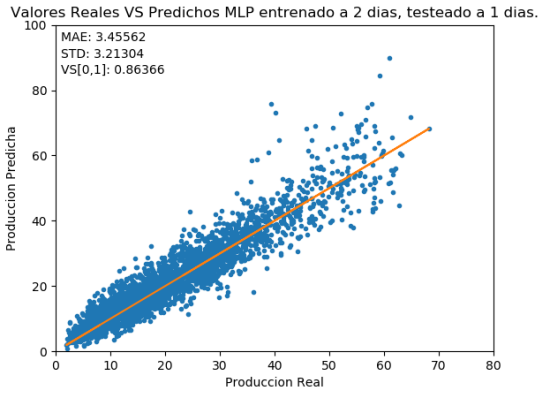


(b) Modelo más eficiente

Figura A.4: Comparativa de test para horizonte de 0 días en el modelo MLP de 2 capas ocultas con 100 neuronas cada una.

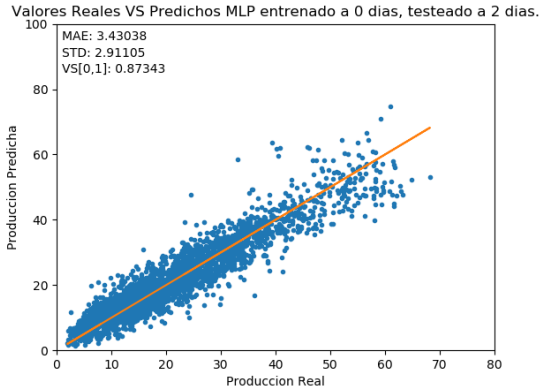


(a) Modelo más reciente

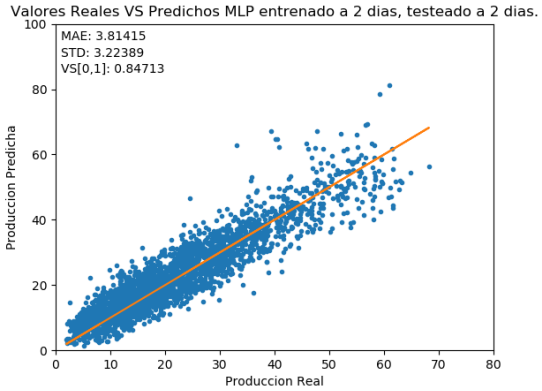


(b) Modelo más eficiente

Figura A.5: Comparativa de test para horizonte de 1 día en el modelo MLP de 2 capas ocultas con 100 neuronas cada una.



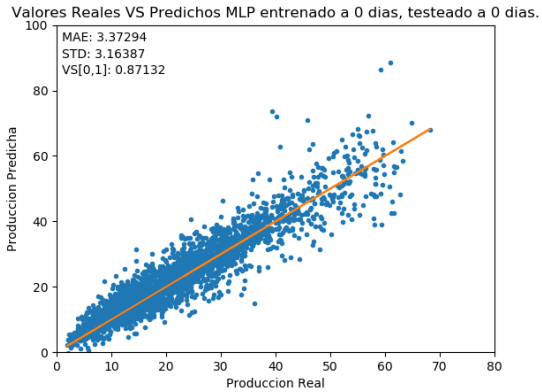
(a) Modelo más eficiente



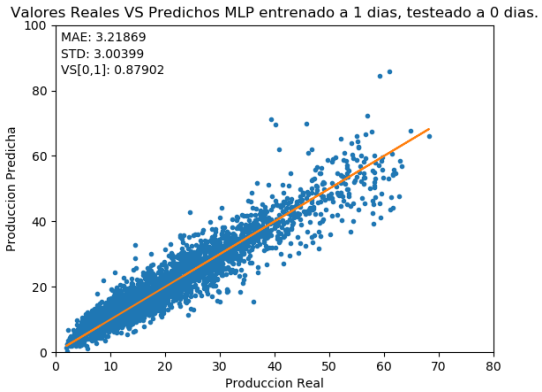
(b) Segundo modelo más eficiente

Figura A.6: Comparativa de test para horizonte de 2 días en el modelo MLP de 2 capas ocultas con 100 neuronas cada una.

A.3. Modelo *MLP* con dos capas ocultas de doscientas neuronas cada una.

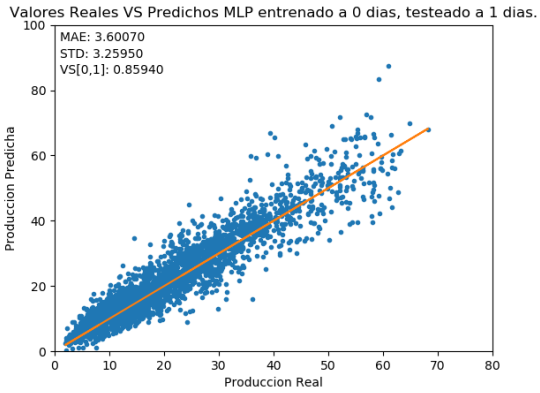


(a) Modelo más reciente

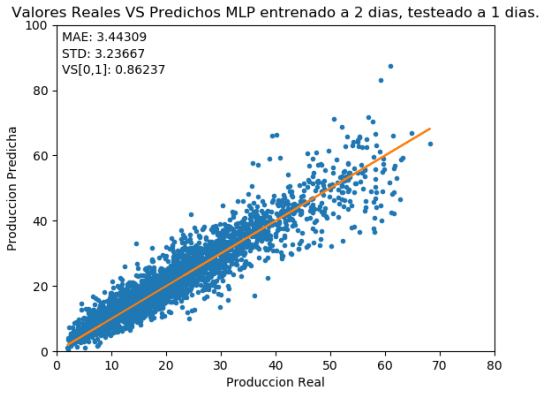


(b) Modelo más eficiente

Figura A.7: Comparativa de test para horizonte de 0 días en el modelo MLP de 2 capas ocultas con 200 neuronas cada una.

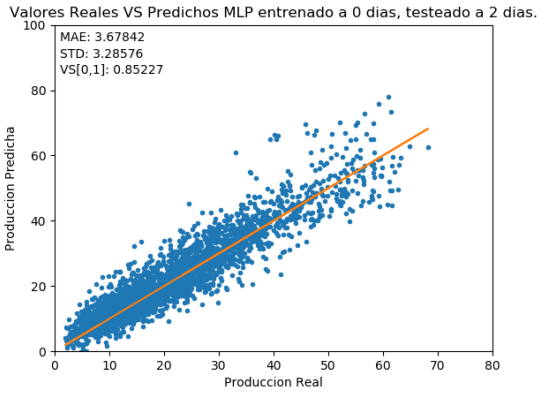


(a) Modelo más reciente

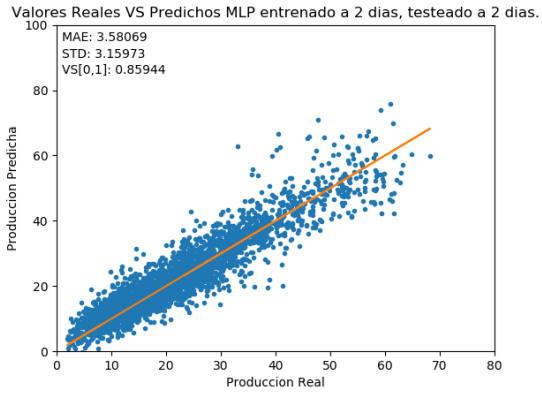


(b) Modelo más eficiente

Figura A.8: Comparativa de test para horizonte de 1 día en el modelo MLP de 2 capas ocultas con 200 neuronas cada una.



(a) Modelo más reciente



(b) Modelo más eficiente

Figura A.9: Comparativa de test para horizonte de 2 días en el modelo MLP de 2 capas ocultas con 200 neuronas cada una.

B

Documentación de los scripts

A continuación se ofrece la documentación generada de todos los *scripts* utilizados para la realización del presente *Trabajo de Fin de Grado*.

B.1. buildDataframes.py

Módulo encargado de la creación de los objetos Dataframes a partir de los archivos GRIB

B.1.1. Documentación de funciones

build_dataframes(fromDir="./GribsPredicciones", initDate="20150101", finDate="20171231", toPath="./DataFrames/", resList=[0.125])

Crea los dataframes diarios del rango de fechas indicado por parámetros y a cada resolución indicada por parámetro.

fromDir: (Default: "./GribsPredicciones") Directorio donde se encuentran los archivos GRIB de predicción meteorológica.

initDate: (Default: "20150101") Fecha de inicio.

finDate: (Default: "20171231") Fecha de fin.

toPath: (Default: "./DataFrames/") Ruta de descarga de los archivos resultantes.

resList: (Default: [0.125, 0.25, 0.5, 1.0]) Lista de resoluciones a las que se quieren obtener los datos.

filtrar_df_res(df, res)

Filtra las componentes de un Dataframe según la resolución pasada por parámetro.

df: dataframe a filtrar

res: resolución a la que se quiere filtrar

return: dataframe filtrado

grib_to_df(grib_file=None, output="./", days=None, resList=[])

FUNCIÓN MODIFICADA A PARTIR DEL CÓDIGO DE JUAN BELLA. Introduce los datos de un fichero grib en un dataframe a distintas resoluciones y lo descarga en la ruta especificada por parámetros. Los datos de la medición se introducen en un dataframe con la fecha de la predicción como índice.

grib_file : nombre del fichero del que tomar los datos

output: ruta en la que guardar el dataframe creado

days: número de días a incluir en el diccionario. None → todos los días (valor por defecto).

resList: (Default: []) Lista de resoluciones a las que se quiere obtener cada dataframe en cuestión.

grib_to_dicc(grib, days=None)

FUNCIÓN MODIFICADA A PARTIR DEL CÓDIGO DE JUAN BELLA. Introduce los datos de un fichero grib en un diccionario. Los datos de la medición se introducen en un diccionario ordenado por la fecha de la toma de datos, fecha de la predicción y nombre de la variable y coordenadas de dónde se ha tomado.

grib: objeto grib que contiene los datos del fichero.

days: número de días a incluir en el diccionario. None → todos los días (valor por defecto).

:return: diccionario con los datos

B.2. buildDaysMatrix.py

Módulo encargado de la creación de las distintas matrices de horizontes temporales.

B.2.1. Documentación de funciones

buildDaysMatrix(days=3, dirDFs="./DataFrames", toPath="./Pickles/", resList=[0.125])

Crea matrices de datos a distintos días de diferencia entre la fecha de análisis la de validez de la predicción según los archivos guardados previamente en Dataframes en el directorio pasado como parámetro para descargar el resultado en el path seleccionado. Adicionalmente se introducirá una lista de resoluciones de mayor a menor resolución según cuántas se quieran obtener.

days: (Default: 3) Rango de días hasta el que quieres llegar, sin incluir, en la creación de

distintas matrices.

dirDFs: (Default: ’./DataFrames’) Directorio en el que se encuentran los Dataframes de los datos diarios.

toPath: (Default: ’./Pickles/’) Path en el que se descargarán los archivos resultantes del proceso

resList: (Default: [0.125, 0.25, 0.5, 1.0]) Lista de resoluciones distintas de archivos que se quieren obtener del proceso.

crear_matriz_datos(days, inputfiles, output, res)

FUNCIÓN MODIFICADA A PARTIR DEL CÓDIGO DE JUAN BELLA. Concatena los ficheros pasados como argumento para crear una matriz de datos. A partir de los ficheros pasados, crea una matriz de datos en la que las fechas de generación y predicción tienen una diferencia de los días pasados como argumento. Adicionalmente se ha añadido el parámetro res para poder seleccionar la resolución a la que se quiere dicha matriz.

days: Número de días entre las fechas de generación y predicción

inputfiles: Lista con la ruta a los ficheros a concatenar

output: Ruta de descarga de los datos a obtener

res: Resolución a la que se quiere crear la matriz

return: Dataframe con los datos obtenidos

B.3. merge.py

Módulo utilizado en la unión de los datos de predicción metorológica con las producciones de energía eólica.

B.3.1. Documentación de funciones

cargar_produciones(file="./Producciones/producciones.csv", zone=.Europe/Madrid")

Se encarga de cargar las producciones a un Dataframe y convertir las horas de las mismas a formato UTC según la zona horaria especificada por parámetros.

file: (Default: "./Producciones/producciones.csv") Archivo CSV con las producciones.

zone: (Default: ’Europe/Madrid’) Zona horaria en la que se encuentran los datos de las producciones.

return: Dataframe de producciones indexado por fecha y hora de produccion en formato UTC.

merge_predicciones_producciones(produccionesDF, prediccionesDir="./Pickles", outsPath="./CompleteDFs

Crea el dataframe completo para cada uno de los archivos del directorio especificado por parámetros y los descarga en la ruta especificada por parámetros.

produccionesDF: Dataframe de las producciones de energía eólica.

prediccionesDir: (Default: "./Pickles") Directorio que contiene as matrices de datos a varios días vista y a distintas resoluciones.

outsPath: (Default: "./CompleteDFs/") Ruta de descarga de los resultados.

tryconvert(date, local)

Función auxiliar para paralelizar la conversión horaria.

date: fecha a convertir.

local: objeto de la librería Pytz correspondiente a la zona de huos horaria de la que se está hablando.

:return: None en caso de que haya habido algún problema en la conversión horaria, hora en UTC si no la ha habido.

B.4. hyper.py

Módulo encargado de la hiperparametrización de los modelos propuestos.

B.4.1. Documentación de funciones

hiperparametrizar(mlps=True, ridges=True, hidden_layer_sizes=(100, 100), dfFolder="./CompleteDFs", days=0, res=1.0, n_folds=4, outPathHyp="./Hyperparametrizations/", trainYearList=[2015,2016)

Hiperparametriza el valor alpha entre las potencias de 10 entre -7 y 6, valor responsable de la regularización de los modelos especificados por parámetros.Descarga los datos de la hiperparametrización de los modelos la ruta especificada por parámetros.

mlps: (Default: True) Especifica si hiperparametrizar modelos MLP.

ridges: (Default: True) Especifica si hiperparametrizar modelos Ridge.

hidden_layer_sizes: (Default: (100, 100)) Tupla especificando la topología del MLP.

dfFolder: (Default: "./CompleteDFs") directrio donde se encuentran los Dataframes comple-

tos.

days: (Default: 0) Días vista con los que entrenar y validar al modelo

res : (Default: 1.0) Resolución de los datos con los que se quiere trabajar.

n_folds : (Default: 4) Número de folds para la cross validation

outPathHyp: (Default: "./Hyperparametrizations/") Ruta de descarga para los datos del prodeco de hiperparametrización.

trainYearList: (Default: [2015, 2016]) Lista de años que se utilizarán para el entrenamiento y la validación de los modelos.

return: Devuelve el mejor modelo en el caso de que se decida hiperparametrizar un modelo mlp.

B.5. scores.py

Módulo utilizado en la obtención de los rendimientos de los modelos previamente hiperparmetrizados.

B.5.1. Documentación de funciones

scores(ridges=True, mlps=True, dirDFs=./CompleteDFs, outsPath="./Scores/", modelsPath="./Hyperparametr
day=0, testDay=0, res=1.0, hidden_layer_sizes=(100, 100), testYearsList=[2017], numModels=5)

Obtiene los resultados de test de los modelos especificados por parámetros utilizando los datos que se encuentren en el directorio indicado de dataframes y los datos de los modelos la ruta de los archivos de hiperparametrización. La especificación del modelo y del test se hace a través de los parámetros que ofrece el método.

ridges: (Default: True) Especifica si se desean testear modelos Ridge

mlps: (Default: True) Especifica si se desean testear modelos MLP

dirDFs: (Default: "./CompleteDFs") Directorio donde se encuentran los Dataframes con los datos meteorológicos con las producciones.

outsPath: (Default: "./Scores/") Ruta de descarga de los gráficos generados.

modelsPath: (Default: "./Hyperparametrizations/") Ruta donde se encuentran los datos de las hiperparametrizaciones de los distintos modelos.

day: (Default: 0) Número de días vista de la predicción meteorológica a los que se ha hiperparametrizado y se entrenará el modelo en cuestión.

testDay: (Default: 0) Número de días vista de la predicción meteorológica a los que se some-

terá el test de cada modelo.

res: (Default: 1.0) Resolución de los datos meteorológicos a la que se trabajará.

hidden_layer_sizes: (Default: (100,100)) Tupla que representa la topología de la arquitectura de red con la que se trabajará.

testYearsList: (Default: [2017]) Lista de años que se utilizarán como test.

numModels: (Default: 5) Número de modelos que se entrenarán y testearán para posteriormente promediar los resultados.

return: Dataframe o dataframes (según número de modelos indicado) con los datos obtenidos.